Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL FINAL DE GRAU

TÍTOL DEL TFG: "MODELING AND IMPLEMENTATION OF SPACECRAFT ATTITUDE DISTURBANCES AND CONTROL SYSTEMS IN TO A SATELLITE SIMULATOR"

TITULACIÓ: Grau en Enginyeria d'Aeronavegació

AUTOR: Juan David Rincón Ortiz

DIRECTOR: PhD Hyuk Park

DATA: October 2nd 2018

Títol: “MODELING AND IMPLEMENTATION OF SPACECRAFT ATTITUDE DISTURBANCES AND CONTROL SYSTEMS IN TO A SATELLITE SIMULATOR”

Autor: Juan David Rincón Ortiz

Director: PhD Hyuk Park

Data: October 2nd 2018

Resum

Aquest treball tracta de crear un simulador de “attitude” per poder estudiar el comportament dels nano-satèl·lits, específicament els CubeSats d’una unitat (1U CubeSats, que tenen com a dimensions 10 x 10 x 10 cm) en orbites de baixa altura (LEO).

En aquest simulador, es té en compte les pertorbacions i els moments externs creats per l’entorn com la pertorbació gravitacional per l’aixafament de la Terra, com el moment creat per la pressió per radiació solar i el moment creat per el camp magnètic de la Terra, entre altres.

El simulador, utilitza les equacions dinàmiques d’Euler amb la finalitat d’obtenir les velocitats angulars creades a partir dels moments externs que actuen sobre el satèl·lit. Una vegada obtingudes les velocitats angulars totals en els tres eixos, fem servir els angles d’Euler per tenir una representació de l’attitude del satèl·lit, aquests angles són el “roll”, “pitch” i “yaw”.

Les equacions han sigut discretitzades fent servir el mètode de Runge-Kutta d’ordre 4 per disminuir l’error i que hi hagi precisió en les simulacions.

Es demostra que la major influença sobre el CubeSat a orbites de baixa altura el té el moment creat pel camp magnètic terrestre. A més que el CubeSat arriba a tenir un moviment estable i periòdic passat un temps, donant la possibilitat de exercir sobre ell qualsevol tipus de control actiu, el més comú es el control per “reaction wheels” i “magnetorquers”.

Title: “MODELING AND IMPLEMENTATION OF SPACECRAFT ATTITUDE DISTURBANCES AND CONTROL SYSTEMS IN TO A SATELLITE SIMULATOR”

Author: Juan David Rincón Ortiz

Director: PhD Hyuk Park

Date: October 2nd 2018

Overview

In this project is wanted to create an attitude simulator to see how nanosatellites (particularly 1U CubeSats that are 10 x 10 x 10 cm of size) behave in front of external torques and perturbations, for instance solar pressure radiation torque, Earth oblateness perturbation and residual dipole moment between others, while orbiting in Low Earth Orbits (LEO).

Simulator is based on Euler’s dynamics equations and from these equations is wanted to know the angular velocities that are caused due to external moments that are acting on the satellite.

Euler equations have been numerically discretized by using 4^{th} order Runge-Kutta. It guarantees a low discretization error than using other methods as the Euler that has more error. With this low error, we get high precision on simulations.

Once total angular velocities are known, we use the Euler Angles representation to plot the roll, pitch and yaw because we are interested on seeing how much has our satellite turned.

In simulations, we conclude that magnetic dipole moment is the one that has more impact on LEO missions for CubeSats. Also we see in the part of angles representation that satellite has a stable motion and periodic, this allow the use of active control on CubeSats without difficulty. Any active control is possible but for nano-satellites the most common are reaction wheels or magnetorques.

Table of Contents

CHAPTER 1: INTRODUCTION

Simulation and modelling have become an important tool in the field of satellite dynamics and control. With the pass of time, the annual number of CubeSats, which are the most common used for either academic or investigation purposes, have been increasing since 2000 up to nowadays [1].

Organizations like ESA, have encouraged this growth using academic programmes such as the *Fly Your Satellite* programme [2] that allows students to be in touch with a real satellite developing experience. Also CubeSats have started to show an increasing potential for commercial use.

Mainly, this project focuses on demonstrating that once CubeSat is deployed in the orbit, it goes through several rotations about its three axes (whose angles of rotation over these axes are roll, pitch and yaw in the Body Reference Frame) due to the external perturbations and over time see that these rotations become stable, making possible to use active control to correct the satellite orientation by means of actuators like thrusters, magnetic torquers and reaction wheels that later on is going to be explained more detailed.

We have two types of ACDS (Attitude Control and Determination System).
In the one hand exists the active control system that determines its current orientation based on calculations from sensor data, then feeds that into a controller or actuator which moves the satellite towards the correct orientation.
On the other hand, we have passive control that has no control and no sensor feedback, the control is “set” beforehand [3].

However, in our simulation is going to be considered a 1U CubeSat (10 x 10 x 10 cm) turning around the Earth in a LEO orbit, for this reason it’s difficult to add an active control system because of the small size.

In order to carry out this kind of projects successfully, is needed to have an accurate simulator where all perturbations and external torques are taken into account. The total sum vector of these torques are included in the Euler equations and our output is the angular velocity. For this, we are going to use Matlab as a principal tool to make all the computations and once we have obtained the angular velocity we compute how much has moved our satellite by using the Euler angles. It must to be said that these equations are discretized using Runge-Kutta 4th order for the Euler Equations and Euler method for Euler angles equations.

This project is organized in the following manner: Chapter one is called theoretical background, it will contain all the theory part distributed in sections and their formulas, Chapter 2 is called simulation and results, here we are going to add the plots of the simulations and make comments about them and finally, the last part are the conclusions where we are going to make an overall explanation about the project and make conclusions of our results.

CHAPTER 2: THEORETICAL BACKGROUND

In this chapter is going to be explained all the theory part which has been implemented in our programming tool in order to get the results from the simulation.

This part gives information about the mathematical modeling of the orbit and the environment. Giving also, an explanation of the different reference frames that are most common used in attitude modeling and how attitude is represented.
Finally, there is a brief introduction about the sensors and actuators used on-board of the satellite that has an active control.

1. Orbit Modeling

Motion of the satellites around the Earth could not happen without the important Kepler's laws of planetary motion.

Kepler was the first to describe the laws that govern the orbits of the planets, from empirical observations of the movement of Mars supported, in large part, in astronomical observations made by Tycho Brahe. Kepler laws are stated as follows:

First Law: The orbit of each planet is an ellipse, with the Sun in one of the two focus.

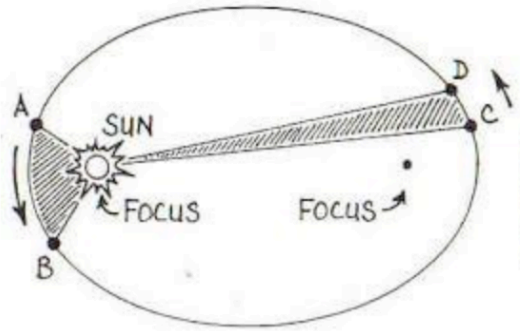

Fig 1.1) Kepler's First Law.

Second Law: The areal velocity of a planet around the sun remains constant. Or, the radius vector drawn from the sun to the planet sweeps out equal areas in equal intervals of time.

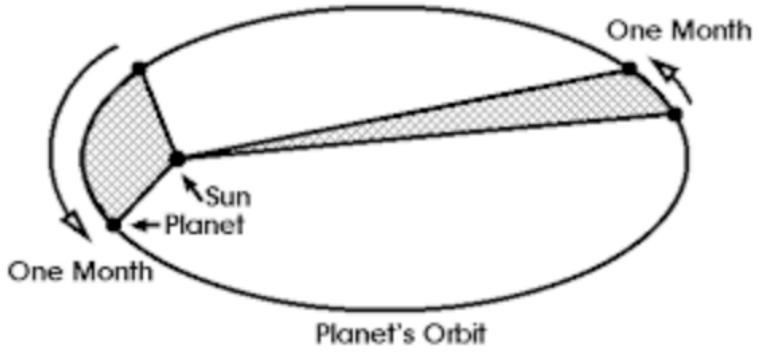

Fig 1.2) Kepler's Second Law.

Third Law: The square of the time period of revolution of a planet around the sun is proportional to the cube of the semi major axis of its elliptical orbit.

$$T^2 \propto a^3 \qquad (1.1)$$

Years later, Newton developed his law of gravitation based in Kepler's work. Isaac Newton introduces the idea that movement of the objects in the sky, such as the planets, the Sun, and the Moon, and the motion of objects around the Earth, like apples that fall from the tree, could be described by the same laws of the physics.

1.1. Orbit Elements

Orbit elements (or Keplerian elements) are the parameters needed to define an orbit. There are six parameters which are:

I. **Semimajor axis (a):** In geometry, the semimajor axis of an ellipse is the half of the larger diameter. In astronomy, it is equivalent to average distance of an object that orbits around another if the central object occupies one of the focus. The semimajor axis is one of the most important features of an orbit as well as its period. It can be mathematically proven that for a body orbiting, the semimajor axis represents the average distance from the body to the gravitational central source.

II. **Eccentricity (e):** Is a parameter that determines the degree of deviation of an orbit compare it to a circle, in other words, it determines its shape.

For a circular orbit the eccentricity is e = 0, and for an extremely elliptical orbit is e = 1. Satellites usually are in orbits between $0 \leq e < 1$.

III. **Inclination (i):** It is the angle that the plane of the orbit (orbital plane) of a star or satellite forms with a plane of reference. The inclination of the orbit is always measured with respect to the equatorial plane of the planet or the body from which they orbit. The equatorial plane is the plane perpendicular to the axis of rotation of the planet and passing through the center of the body. In this way, they can be classified in these cases:

- An inclination of 0 ° means that the orbital body is orbiting in the plane of the equator of the planet, and it is rotating in the same direction as the planet.

- An inclination of 90 ° indicates that the orbital body is in a polar orbit, in this case the object passes through the poles (north and south) of the planet in successive turns.

-An inclination of 180 ° indicates that the orbital object is making a retrograde movement in the equatorial plane of the orbit.

IV. **Longitude of the ascending node (Ω):** It is the angle expressed in degrees that forms the Earth radius that passes through the ascending node of a determined orbit, and the "vernal" point (Aries). Aries, is a point of reference of the celestial sphere, to which all the straight ascensions of the stars refer and is defined as the point of intersection of the ecliptic (curvilinear path that apparently describes the Sun around the Earth) with the terrestrial equatorial plane in the spring equinoccio (ascending direction).

V. **Argument of periapsis (ω):** Is the angle (in degrees) formed between the line that is projected from the center of the earth to the satellite when it cuts the equator in its ascending orbit.

VI. **Mean anomaly (M):** Is the angle that is projected uniformly in time, between 0 and 360 degrees during a revolution (orbit), establishing 0 degrees for the perigee and 180 degrees for the apogee. Fixes the position of the satellite within an orbit.

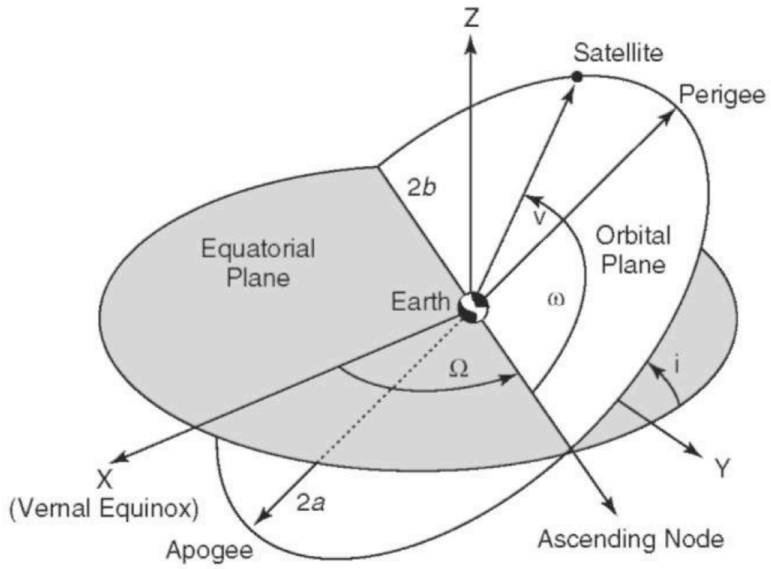

Fig 1.3) Keplerian elements.

1.2. Orbit Propagator

Propagation refers to the determination of the motion of a body over time. According to Newton's laws, the motion of a body depends on its initial state (that is, its position and orientation at some known epoch) and the forces acting on it over time.

It is known that the Earth is not spherically perfect because the mass is distributed unevenly within the planet. **Spherical harmonic coefficients for the potential** represents this mass distribution of the Earth. The most important harmonic coefficients are:

$J_2 = 1.083 \cdot 10^{-3}$

$J_3 = -2.534 \cdot 10^{-6}$

$J_4 = -1.620 \cdot 10^{-6}$

$J_5 = -2.273 \cdot 10^{-7}$

The meaning of the different harmonic coefficients is showed in the following image (Fig 1.4), where the first figure from the top corresponds to J_2, the second figure from the top corresponds to J_3, J_4 and J_5 are the two figures from the bottom respectively.

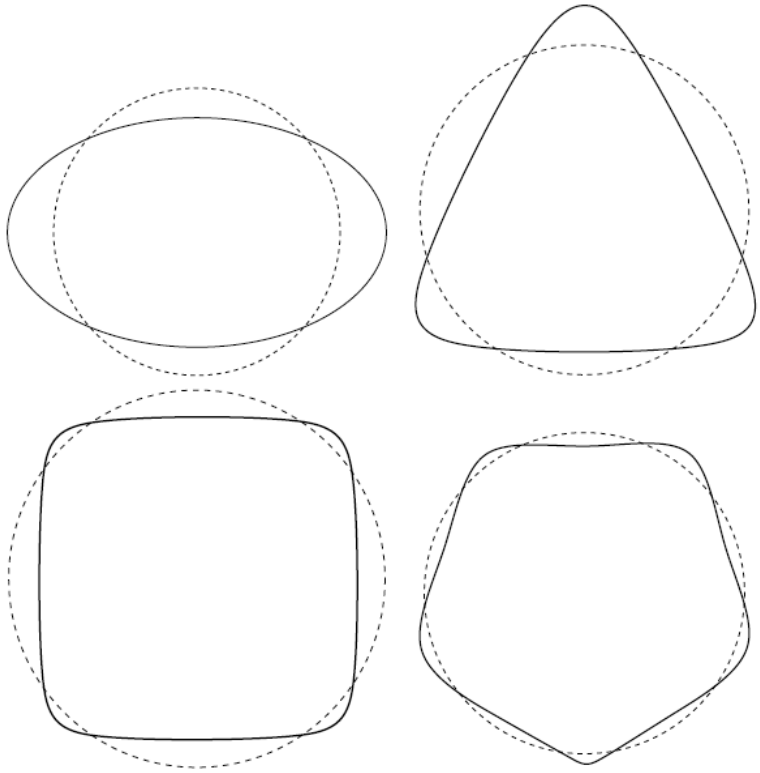

Fig 1.4) Representation of harmonic coefficients.

CubeSats are usually positioned in LEO orbits (between 200 - 2000 km of altitude) where the most influence effect over small satellites is the J_2. So it is common to neglect the effect from the other harmonics and consider the Earth as an oblate spheroid.

Gravitational potential for a non-spherical Earth can be computed as,

$$U = \frac{\mu}{r} \cdot \left[1 - \sum_{n=2}^{\infty} J_n \cdot \left(\frac{R}{r}\right)^2 \cdot p_n\big(sin(\varphi)\big) \right] \quad \textbf{(1.2)}$$

where U is Newton's 2-body gravity term, μ = GM is Earth's gravitational constant, R is Earth's equatorial radius, p_n are Legendre polynomials, φ is geocentric latitude, and J_n are dimensionless geopotential coefficients.

Considering only the effect of J_2 (from eq. 1.2), the final gravitational potential is:

$$U = \frac{\mu}{r} \cdot \left[1 + \frac{J_2}{2} \cdot \left(\frac{R}{r}\right)^2 \cdot \big(1 - 3 \cdot sin^2(\varphi)\big) \right] \quad \textbf{(1.3)}$$

In absence of perturbations, the simplest propagator is the keplerian, but we are trying to develop a realistic propagator were most of the important perturbations are taken into account.

Once keplerian elements have been introduced and has been developed a formula for the perturbation due to the oblateness of the Earth, now we can define our propagator [9][10]:

$$n = \sqrt{\frac{G \cdot M}{a^3}} \quad \textbf{(1.4)}$$

$$\Omega = \boldsymbol{\Omega_o} - \frac{\mathbf{3}}{\mathbf{2}} \cdot \boldsymbol{n} \cdot \left(\frac{R}{p}\right)^2 \cdot J_2 \cdot cos(i) \cdot dt \quad \textbf{(1.5)}$$

$$\omega = \omega_o + \frac{3}{4} \cdot n \cdot \left(\frac{R}{p}\right)^2 \cdot J_2 \cdot (5 \cdot cos^2(i) - 1) \cdot dt \quad \textbf{(1.6)}$$

$$M_k = Mo + \left(n + \frac{3}{4} \cdot n \cdot \left(\frac{R}{p}\right)^2 \cdot J_2 \cdot \sqrt{1 - e^2} \cdot (2 - 3 \cdot sin^2(i))\right) \cdot \boldsymbol{dt} \quad \textbf{(1.7)}$$

$$E_k = M_k + e \cdot sin(E_k) \quad \textbf{(1.8)}$$

Where E_K (0) = M_k , we have to compute repeatedly,

$$E_k(n) = M_k - e \cdot sin(E_k(n-1)) \quad \textbf{(1.9)}$$

until,

$$|E_k(n) - E_k(n-1)| < 10^{-8} \quad \textbf{(1.10)}$$

In this step we are going to compute the **true anomaly (v)** that is an angular parameter that defines the position of a body moving along a Keplerian orbit. It is the angle between the direction of periapsis and the current position of the body, as seen from the main focus of the ellipse (the point around which the object orbits).

$$sin(v_k) = \frac{\sqrt{1-e^2} \cdot sin(E_k)}{1 - e \cdot cos(E_k)} \quad \textbf{(1.11)}$$

$$cos(v_k) = \frac{cos(Ek\ ) - e}{1 - e \cdot cos(E_k)} \quad \textbf{(1.12)}$$

From equations 1.11 and 1.12, to extract the true anomaly, we can use the complex number functions. In other words, converting the sinus and cosinus of the true anomaly into a complex number and then compute the argument with the help of Matlab.

Also it is needed to compute the radius of the orbit, if the orbit is a circle, the radius will be constant, but in case that is an ellipse, the radius will vary.

$$r_k = a \cdot (1 - e \cdot cos(E_k)) \quad \textbf{(1.13)}$$

Specific angular momentum (h) will be used when we convert from keplerian orbital elements to ECI coordinates.

$$h = \sqrt{\mu \cdot a \cdot (1 - e^2)} \quad \textbf{(1.14)}$$

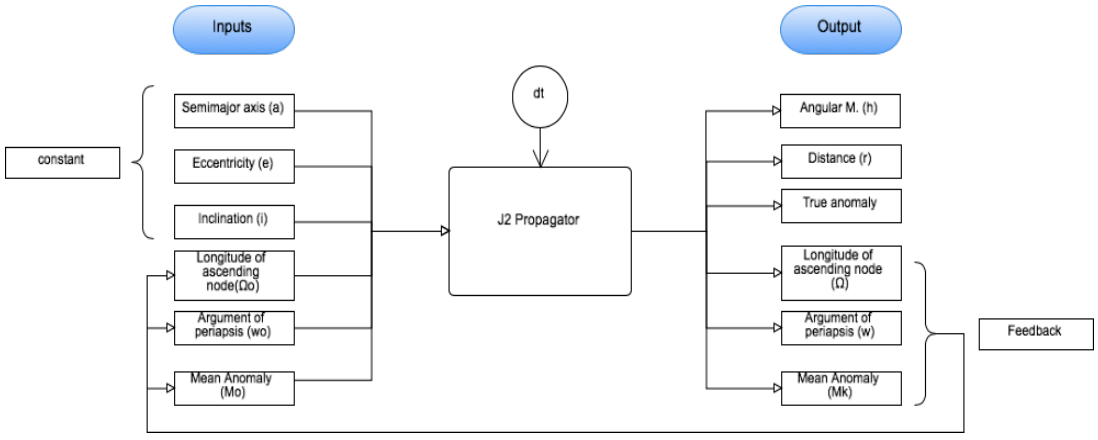

Fig 1.5) Orbital Propagator Flowchart.

Semimajor axis, eccentricity and orbital inclination will be constant along the propagation, the terms that change in each iteration are the longitude of

ascending node, argument of periapsis and the mean anomaly due to the effect of J_2 (in case of the simplest keplerian propagator, the argument of periapsis and the longitude of ascending node are constant as well).

Figure 1.5 shows how is distributed our propagator in order to implement it in Matlab.

Finally, we are going to convert the position and its velocity of our CubeSat from Keplerian elements to ECI coordinates. ECI conversion will be useful for the computation of external torques and perturbations.

Position:

$$X = r \cdot \left(cos(\Omega) \cdot cos(\omega + v) - sin(\Omega) \cdot sin(\omega + v) \cdot cos(i)\right) \quad \textbf{(1.15)}$$

$$Y = r \cdot \left(sin(\Omega) \cdot cos(\omega + v) + cos(\Omega) \cdot sin(\omega + v) \cdot cos(i)\right) \quad \textbf{(1.16)}$$

$$Z = r \cdot \left(sin(\omega + v) \cdot sin(i)\right) \quad \textbf{(1.17)}$$

Velocity:

$$V_X = \frac{X \cdot \hbar \cdot e}{r \cdot p} \cdot sin(v) - \frac{\boldsymbol{h}}{\boldsymbol{r}} \cdot \left(cos(\Omega) \cdot sin(\omega + v) + sin(\Omega) \cdot cos(\omega + v) \cdot cos(i)\right) \quad \textbf{(1.18)}$$

$$V_Y = \frac{Y \cdot \hbar \cdot e}{r \cdot p} \cdot sin(v) - \frac{\boldsymbol{h}}{\boldsymbol{r}} \cdot \left(cos(\Omega) \cdot sin(\omega + v) + sin(\Omega) \cdot cos(\omega + v) \cdot cos(i)\right) \quad \textbf{(1.19)}$$

$$V_Z = \frac{Z \cdot \hbar \cdot e}{r \cdot p} \cdot sin(v) + \frac{\boldsymbol{h}}{\boldsymbol{r}} \cdot \left(sin(i) \cdot cos(\omega + v)\right) \quad \textbf{(1.20)}$$

2. Reference Frames

It is important to know which are the different reference frames for representing satellite's position and attitude.

A reference frame consists of three perpendicular unit vectors. References frames are categorized according to the origin locations (intersection of three orthogonal vectors). It can be categorized as: Geocentric, Topocentric, Selenocentric, Heliocentric, Geodetic, Orbital and Body reference frames [4].

Reference frames can be inertial or non-inertial. Newton's laws of motion give the knowledge to define which frame is inertial or not. First law which is the law of inertia states that a body stays at rest or continues to its uniform motion unless there is an external force applied on it [5]. Law of force which is the second law defines that a force acting on a body makes a body accelerate in the direction of the force proportional to its mass [5]. If the Newton's laws above are applicable for a frame, it is known as Newtonian or inertial reference frame. Since the mathematical models of sensors, orbit model and sensors are in different reference frames, they have to be defined accurately.

2.1. Earth-Centered Inertial (ECI) Frame

ECI frame is an inertial frame that it is fixed in space, so it is a non-accelerated reference frame in which Newton's Laws of motion are valid. The origin of the frame is placed at the centre of the Earth; the x-axis is pointing towards the Vernal Equinox, that is the intersection of the Earth's equatorial plane with the plane of the Earth's orbit around the Sun. The z-axis points to the North pole and the y-axis completes the right hand Cartesian coordinate system. Velocity of the orbit frame and the motion of the Sun can be directly compared to this frame as well as all different satellite motions can be presented in this reference frame. This frame is denoted by "I".

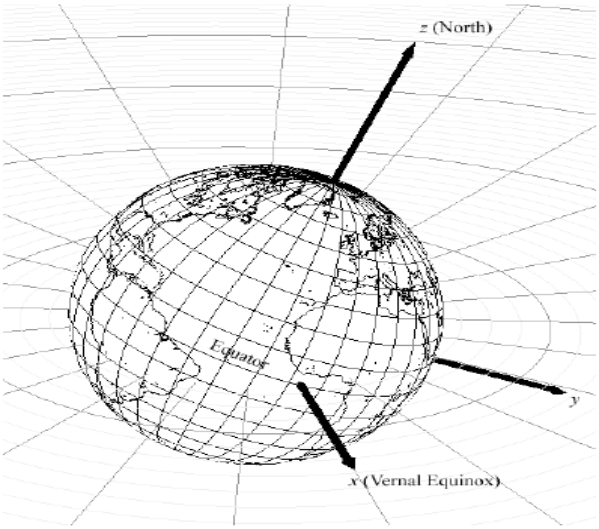

Fig 1.6) ECI Frame [6].

2.2. Earth-Centered Earth-Fixed (ECEF) Frame

The origin of this frame is located at the center of the Earth. The x-axis points to the prime meridian and y-axes (that completes the right hand Cartesian coordinate system) rotate about the z-axis relative to the ECI frame.

The angular velocity of the Earth is $7.2921 \cdot 10^{-5} \frac{rad}{s}$.

Magnetic field around the Earth, IGRF, can be used with an orbit estimator to create a reference model by the help of this frame. The frame is denoted by “E”.

2.3. Orbit Plane Reference frame

The **Keplerian elements** are given in this frame. The x-axis points toward the perigee, y-axis along the semiminor axis and z-axis is perpendicular to the plane. The frame is denoted as OC. We will need this frame in order to create our orbital **propagator.** Also is needed to make computations that are required to be in other reference frames as ECI or Body frame by using rotation matrices. The orbit plane frame is denoted as “OP”.

2.4. Local Orbit Reference Frame

Orbit frame (or Local Vertical Local Horizontal frame) rotates relative to the ECI frame, with a rate of ω_o depending on the **altitude** of the satellite. The origin is at the center of mass of the satellite. The x-axis (O_1) is toward the direction of motion tangentially to the orbit. The tangent is only perpendicular to the radius

vector in circular orbit, not align with the velocity vector of the satellite in elliptical orbits. The z-axis (O_3) points toward the center of Earth (or nadir), and the y-axis (O_2) completes the right hand system. The local orbit frame is denoted as “O”.

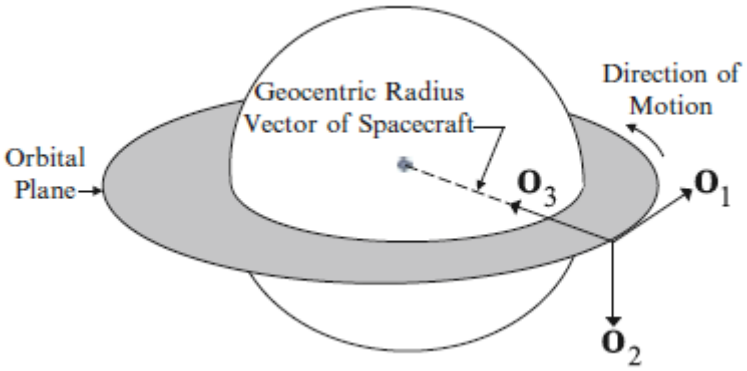

Fig 1.7) Orbital Frame [7].

2.5. Body Reference Frame

Body reference frame starts off from the center of mass of the satellite, and the axes can be modelled as the user prefers but is common to model the axes like: the x-axis following the orbital trajectory, z-axis points to nadir and y-axis completes the right hand orthogonal system. The orientation of the satellite is determined relative to the Orbit frame, while angular velocities are expressed in Body frame. The body frame is denoted as “B”.

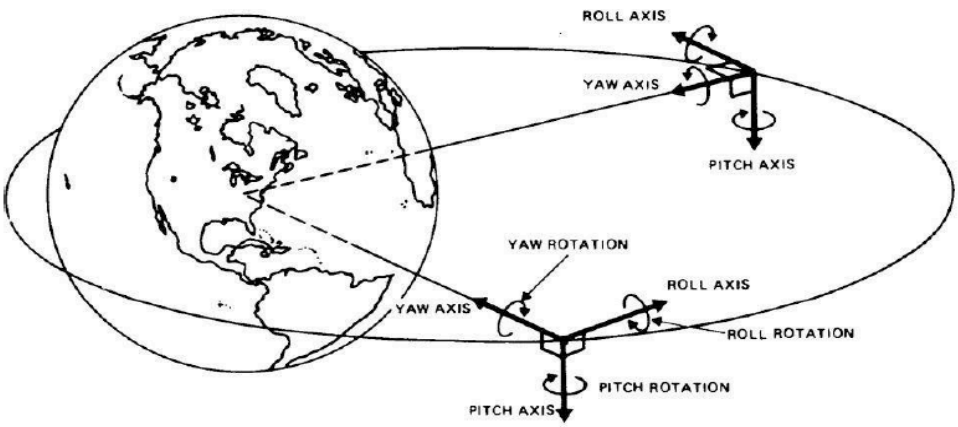

Fig 1.8) Body Reference Frame [8].

3. Attitude Representations

Satellite attitude is referenced to Earth-Fixed reference frame in order to obtain angle data from angular velocity in body fixed frames, so a conversion method is needed to demonstrate the velocity vector correctly. Euler angle transformation is one of the methods applied in transformations. Although this method is reliable, simple and accurate enough for applications, sometimes singularities occur in coordinate transformations. In order to avoid singularities, Quaternion representation can be used as representation method. Also it increases the computation speed, which is critical for navigation applications. On the other hand, it's sometimes hard to visualise the actual angles.

There are many other ways to represent attitude as Rodrigues Parameter representation and the modified Rodrigues Parameters [7], but in this section we are going to focus in Euler angle representation and Quaternions.

3.1. Euler Representation

Euler reasoned that any rotation from one frame to another can be visualized as a sequence of three rotations about base vectors [12].

Euler angle transformation can be presented by using roll, pitch, yaw angles. These angles help to determine the attitude of the satellite relative to the Orbit frame. The roll angle (ϕ) is a rotation angle about the xo-axis, the pitch angle (θ) a rotation angle about the yo-axis and lastly, the yaw angle (ψ) is a rotation angle about the zo-axis.

In this project we are going to use this type of representation due to the easiest way to visualize rotations about the three axis.

3.2. Quaternions

Quaternion representation expresses the attitude matrix as a homogenous quadratic function of the elements of the quaternion, requiring no trigonometric or other transcendental function evaluations. Quaternions are more efficient for specifying rotations than the attitude matrix itself, having only four components instead of nine, and obeying only one constraint, the norm constraint, instead of the six constraints imposed on the attitude matrix by orthogonality [7].

Quaternion (q) is a complex number with one real part (γ) and three imaginary parts (ε):

$$\gamma = cos\frac{\theta}{2} \quad \textbf{(1.21)}$$

$$\varepsilon = [\varepsilon_1\ \varepsilon_2\ \varepsilon_3] = \tau \cdot sin\frac{\theta}{2} \quad \textbf{(1.22)}$$

$$q = [\gamma\ \varepsilon_1\ \varepsilon_2\ \varepsilon_3]^T \quad \textbf{(1.23)}$$

Where θ is the rotation angle about the unit vector τ. Quaternions satisfy:

$$\gamma^2 + \varepsilon_1^2 + \varepsilon_2^2 + \varepsilon_3^2 = 1 \quad \textbf{(1.24)}$$

4. Transformations

Once reference frames have been explained and what types of frames exist, is important to know how could we transform from a reference frame to another. In our case it's important to know some vectors as torques or forces that must be in the body frame (in order to apply the sum of the whole vectors acting upon the satellite), but those vector are represented in other reference frames. Then, in this section we are going to write the different cases of transformation matrices that have been used later on for the simulator.

The rotation matrix can behave as a transformation of a vector represented in one coordinate frame to another frame, as a rotation of a vector within the same frame and finally as a description of mutual orientation between two frames.

The rotation matrix *R* from frame *a* to *b* is denoted by R_a^b. So the rotation of a vector from one frame is written by the following notation:

$$v^{to} = R_{from}^{to} \cdot v^{from} \quad \textbf{(1.25)}$$

4.1. Transformation from ECI to Orbit Plane R.F

The transformation from the Inertial Reference Frame to the Orbit Plane is done by the following rotation matrix [11]:

$$R_{ECI}^{OP} = \begin{vmatrix} \cos(\Omega) & -\sin(\Omega)\cdot\cos(i) & \sin(\Omega)\cdot\sin(i) \\ \sin(\Omega) & \cos(\Omega)\cdot\cos(i) & -\cos(\Omega)\cdot\sin(i) \\ 0 & \sin(i) & \cos(i) \end{vmatrix} \quad \textbf{(1.26)}$$

Where Ω is the Longitude of the ascending node and i is the orbital inclination. (Parameters have been explained in section **1.1**).

4.2. Transformation from Orbit Plane R.F to Local Orbit R.F

If we want to transform a vector from OP reference frame to Local Orbit reference frame there are two rotations that must be done. The first one has a rotation about x-axis of 90 degrees and a rotation in the y-axis of 90 degrees plus the true anomaly (υ).

$$R_{OP}^{O} = R_x(90^o)\cdot R_y(90^o + \upsilon) \quad \textbf{(1.27)}$$

The rotation matrix is [11]:

$$R_{OP}^{O} = \begin{vmatrix} -\sin(\upsilon) & 0 & -\sin(\upsilon) \\ \cos(\upsilon) & 0 & \sin(\upsilon) \\ 0 & 1 & 0 \end{vmatrix} \quad \textbf{(1.28)}$$

4.3. Transformation from Local Orbit R.F to Body R.F

In this transformation, Euler angles are taking part of the rotation matrix. So later in the simulation, it is important to update the angles in each iteration. Rotation matrix from local orbit r.f to body r.f is [11]:

$$R_{O}^{B} = R_x(\psi)\cdot R_y(\theta)\cdot R_z(\phi) \quad \textbf{(1.29)}$$

$$R_{O}^{B} = \begin{vmatrix} \cos\psi\cdot\cos\theta & \cos\psi\cdot\sin\theta\cdot\sin\phi - \sin\psi\cdot\cos\phi & \cos\psi\cdot\sin\theta\cdot\cos\phi + \sin\psi\cdot\sin\phi \\ \sin\psi\cdot\cos\theta & \sin\psi\cdot\sin\theta\cdot\sin\phi + \cos\psi\cdot\cos\phi & \sin\psi\cdot\sin\theta\cdot\cos\phi - \cos\psi\cdot\sin\phi) \\ -\sin\theta & \cos\theta\cdot\sin\phi & \cos\theta\cdot\cos\phi \end{vmatrix} \quad \textbf{(1.30)}$$

5. Orbit Environment Modeling

This section gives information about how the environment of the orbit has been modelled, including Earth's magnetic field, sun position and eclipse models are presented.

5.1. Sun position model

If the direction of the sun is known, it provides a well-defined reference vector that can be used in the satellite's attitude estimation. This vector is also required to estimate solar pressure. To be able to estimate the sun's direction vector, the relationship between the Earth and the Sun has to be known [13]. We know that the Earth revolves around the Sun but an easy way to understand how to get this vector we are going to imagine that the Sun revolves around the Earth by an imaginary orbit (as illustrated in the fig 1.9) because inertial frames for the satellite motion is Earth-centered.

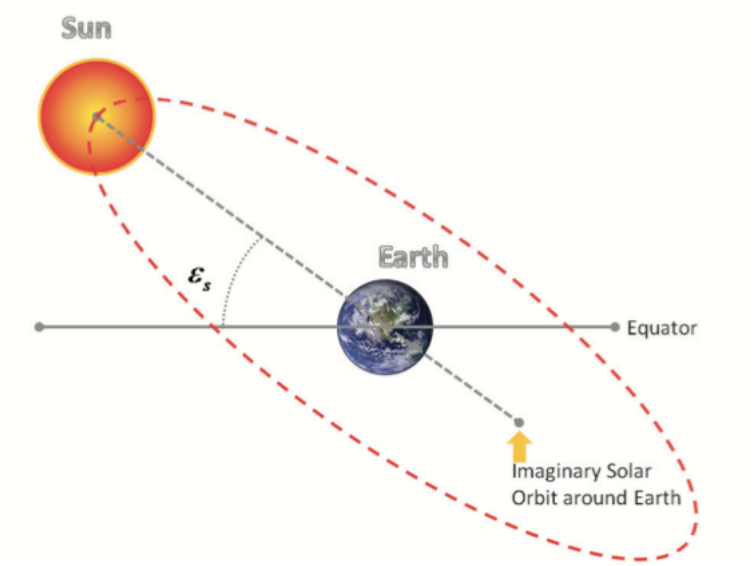

Fig 1.9) Sun elevation in an imaginary orbit around the earth [13].

Kristiansen et al. [14] proposed a formula to compute the solar elevation (ε_s):

$$\varepsilon_S = \frac{23 \cdot \pi}{180} \cdot \sin\left(\frac{2 \cdot \pi}{365} \cdot T_S\right) \quad \textbf{(1.31)}$$

Where T_S is the time elapsed since the first day of spring. The sun's position λ_S (as illustrated in fig 1.10) is given by the following formula:

$$\lambda_S = \frac{2 \cdot \pi}{365} \cdot T_S \tag{1.32}$$

If we know the sun elevation and the sun position in relation to the Earth, now it is possible to compute a vector pointing to the sun. The calculation starts from the initial position given on the first day of spring (vernal equinox) that it is:

$$\hat{S_0}|_I = [1 \quad 0 \quad 0]^T \tag{1.33}$$

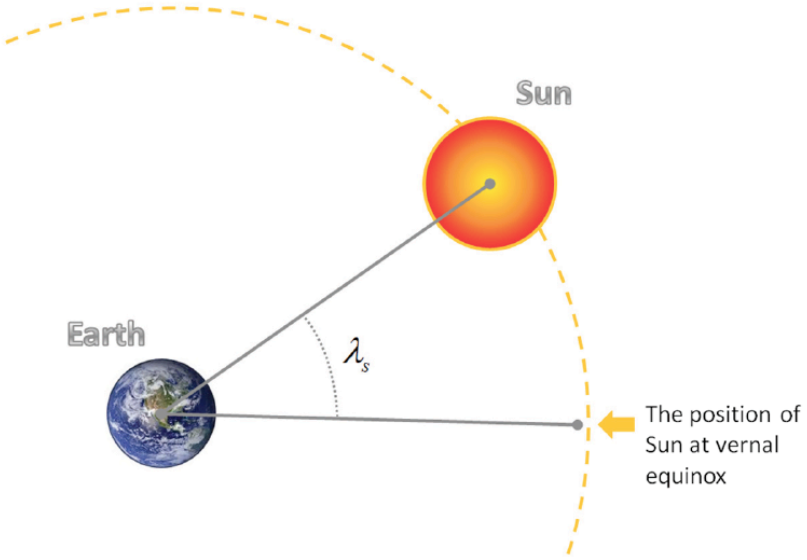

Fig 1.10) Sun position in an imaginary orbit around the earth [13].

The position vector of the sun can be calculated as rotations (in ECI reference Frame):

$$\hat{S}|_{ECI} = \begin{vmatrix} \cos \varepsilon_S \cdot \cos \lambda_S \\ \sin \lambda_S \\ \sin \varepsilon_S \cdot \cos \lambda_S \end{vmatrix} \tag{1.34}$$

5.2. Eclipse model

While orbiting the Earth, the spacecraft may enter eclipse conditions depending on the orbital parameters (depending on the orbit altitude, velocity increases or

decreases and that makes the spacecraft to be more time in eclipse or less time).

There are different ways to model an eclipse, the most common are: the conical shadow model and the cylindrical shadow model.

Conical shadow model is capable to distinguish between umbra (full shadow) and penumbra (partial shadow) conditions. However, this method implies the use of trigonometric functions, which are to be avoided whenever possible due to the high computational expense associated with the evaluation of a truncated polynomial expansion.

Cylindrical shadow model does not require evaluation of trigonometric calculations. This model is not capable to distinguish between umbra and penumbra but as our CubeSat is in LEO orbits (penumbra duration are short on the order of 10 seconds), so cylindrical shadow model is quite accurate for this kind of simulations.

In our simulator we are using cylindrical shadow model, with this model, Earth shadow is simulated as a cylinder of infinite length and radius equal to Earth's radius, R_E. Considering a unit shadow vector $\widehat{u_{sh}}$ antiparallel to sun vector.

$$\hat{u}_{sh} = -\hat{S}|_{ECI} \quad \textbf{(1.35)}$$

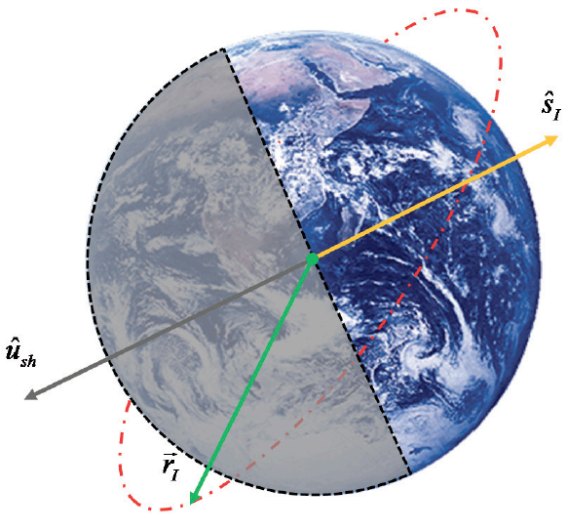

Fig 1.11) Unit eclipse vector [13].

Two conditions tell us if our spacecraft is situated in eclipse:

1) Spacecraft must be located on the night side of the Earth, mathematically is [13]:

$$\hat{u}_{s\hbar} \cdot \hat{r}|_{ECI} > 0 \quad \textbf{(1.36)}$$

2) Spacecraft must be within the shadow cylinder [13]:

$$|R \cdot \|\hat{u}_{s\hbar} \times \hat{r}|_{ECI}\|| \leq R_E \quad \textbf{(1.37)}$$

Where R_E is the equatorial radius of the Earth and $\hat{r}|_{ECI}$ is the unit position vector of the spacecraft.

5.3. Earth's magnetic field model

Earth's magnetic field cannot be neglected since it affects strongly to small satellites as CubeSats and it changes throughout the orbit.

Earth's magnetic field can be computed by the following formula:

$$B = -\nabla \cdot V \quad \textbf{(1.38)}$$

Where B is the Earth's magnetic field, ∇ is the gradient operator and V is the scalar potential function.

IGRF (International Geomagnetic Reference Field) model uses this scalar potential function in order to get the theoretical undisturbed Earth's magnetic field in each point of the surface. It is modelled as:

$$V(r,\theta,\phi) = a \cdot \sum_{n=1}^{\infty} \left(\frac{a}{R}\right)^{n+1} \cdot \sum_{m=0}^{n} P_n^m \cos\theta \cdot (g_n^m \cdot \cos m\phi + \hbar_n^m \cdot \sin m\phi) \quad \textbf{(1.39)}$$

Where $V(r,\theta,\phi)$ is the potential function of the field expressed in spherical harmonics form, *a* is the mean radius of Earth, $P_n^m \cos\theta$ are Schmidt quasinormalized associated Legendre functions of degree *n* and order *m*, g_n^m and $\hbar_n^m$ are the Gaussian constants. ϕ and ϕ are the longitude and colatitude (90º - latitude), respectively [15].

6. Perturbations and Disturbance Moments

Perturbations of the orbit are the results of various forces which are acting on a satellite that perturb it away from the nominal orbit. These perturbations, or variations in the orbital elements, can be classified based on how they affect the Keplerian elements whereas disturbance moments only modify the attitude of the satellite.
The principal sources of perturbations are the gravitational due to the oblateness of the Earth, atmospheric drag and solar pressure radiation.

On the other hand, the forces that also changes the attitude of the satellite by creating a disturbance moment are the gravity gradient torque, solar pressure torque, the aerodynamic drag torque and the residual dipole torque.

In this section we are going to explain only the disturbance moments because are the factors that have been used for the simulator. Perturbations as the oblateness of the Earth has been taken into account while building up our propagator (see section **1.2**) and the other perturbations (solar pressure and aerodynamic drag) are forces that also create a disturbance torque, so they are going to be explained in this section.

6.1. Gravity Gradient Torque

A gravity gradient torque arises from the fact that in a central force field, minuscule accelerations acting on all mass elements of a rigid body are directed towards the center of gravity of the primary body, that is the Earth.

The following model is based on the two-body approximation, it means that lunar and solar gravity sources are neglected. Also, the spacecraft is assumed to be rigid and small compared to its distance from the center of the Earth.

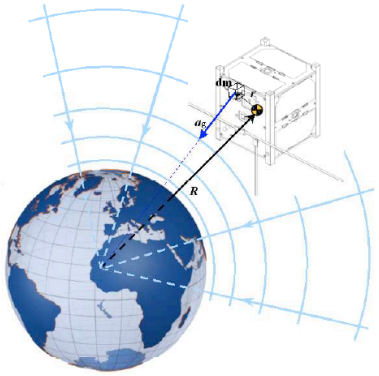

Fig 1.12) Gravity Gradient torque [16].

The gravity gradient torque can be calculated by the following formula [13]:

$$\vec{\tau}_{GG} = 3 \cdot \frac{\mu}{r} \cdot (\vec{u}_{nadir} \times J \cdot \vec{u}_{nadir}) \quad \textbf{(1.40)}$$

Where *J* is the satellite's inertia tensor, $\vec{u}_{nadir}$ is the vector pointing to the center of the Earth (this vector is in the local orbit frame, it must be rotated to the body frame), μ is the standard gravitational parameter and *r* is the altitude of the satellite.

6.2. Solar Radiation Pressure Torque

A satellite in low earth orbit is affected by three major electromagnetic radiation sources in space. These sources are the sun, solar radiation reflected by the Earth (termed albedo), and the thermal infrared radiation of the Earth. Only the first source, direct sun radiation, is modelled as a disturbance forces as it is the largest in magnitude. The sun's electromagnetic radiation exerts a normal force on space objects, known as solar radiation pressure [13].

If there is a non-zero distance between the geometric center and the center of mass, this pressure causes a solar pressure disturbance moment. It can be obtained by the following formula:

$$\vec{F}_{Solar,i} = -P_s \cdot A_{S,i} \cdot \cos\theta_i \cdot \left[(1-\varepsilon) \cdot \hat{s}_{|B} + 2 \cdot \varepsilon \cdot \cos\theta_i \cdot \hat{n}_i|_B\right] \quad \textbf{(1.41)}$$

$$\cos\theta_i = \hat{s}_{|B} \cdot \hat{n}_i|_B \quad \textbf{(1.42)}$$

$$\vec{\tau}_{solar} = \sum_{i=1}^{n} \left(\vec{r}_{GC,i/CoM} \times \vec{F}_{Solar,i}\right) \quad \textbf{(1.43)}$$

Where *n* is the number of satellite surfaces which has the effective angle (0−90◦) with the sun vector, P_s is solar radiation pressure near Earth and its value is 4.56°·10−6 $\frac{N}{m^2}$, A is the exposed area of the satellite to Sun, $\hat{s}_{|B}$is a unit sun direction vector on body frame, $\hat{n}_i|_B$ is the normal vector of exposed ith face on body frame, ε is reflectivity , A_i is the area of the i^{th} surface and $\vec{r}_{GC,i/CoM}$is the vector from the center of mass to the area center of the i^{th} surface.

6.3. Aerodynamic Drag Torque

One of the important factors that affects the satellite is the atmospheric drag, which itself depends on the atmospheric density and the form factor of the object flying into that atmosphere. Drag forces have an effect on a satellite's motion and change the orbit shape as a result of the presence of molecules of gases in the Earth's upper atmosphere.

Many density profile models exist but MSISE-90 [16] is the recommended ECSS (European Cooperation for Space Standardization) standard atmosphere model, from which the main thermodynamic parameters of the atmosphere at 600, 700 and 800km and during low, mean and high solar and geomagnetic activity are showed below.

Activity Level	low		
Altitude [km]	**600**	**700**	**800**
Temperature [K]	699.1631	699.1631	699.1631
Density [kg/m³]	$1.03 \cdot 10^{-14}$	$3.58 \cdot 10^{-15}$	$1.91 \cdot 10^{-15}$
Pressure [N/m²]	$1.09 \cdot 10^{-8}$	$6 \cdot 10^{-9}$	$3.96 \cdot 10^{-9}$
Molecular Weight [kg/mol]	5.5149	3.4648	2.8075
Scale Height [km]	71.0934	129.9408	188.1991

Activity Level	mean		
Altitude [km]	**600**	**700**	**800**
Temperature [K]	1011.533	1011.537	1011.538
Density [kg/m³]	$1.56 \cdot 10^{-13}$	$3.91 \cdot 10^{-14}$	$1.25 \cdot 10^{-14}$
Pressure [N/m²]	$1.01 \cdot 10^{-7}$	$3.36 \cdot 10^{-8}$	$1.58 \cdot 10^{-8}$
Molecular Weight [kg/mol]	13.0389	9.7818	6.6572
Scale Height [km]	68.1361	78.5188	101.1751

Activity Level	extremely high		
Altitude [km]	**600**	**700**	**800**
Temperature [K]	1622.042	1622.087	1622.093
Density [kg/m³]	$6.20 \cdot 10^{-12}$	$2.38 \cdot 10^{-12}$	$9.59 \cdot 10^{-13}$
Pressure [N/m²]	$5.31 \cdot 10^{-6}$	$2.11 \cdot 10^{-6}$	$8.84 \cdot 10^{-7}$
Molecular Weight [kg/mol]	15.7321	15.2723	14.6447
Scale Height [km]	102.6271	108.0038	111.8358

Fig 1.13) Tables from ECSS that show the standard atmospheric density at 600, 700 and 800 km altitude and for different levels of solar activity [16].

The way to obtain the aerodynamic drag torque is quite similar to the solar radiation pressure torque:

$$\vec{F}_{Aero,i} = -\frac{1}{2} \cdot CD \cdot A_{D,i} \cdot \rho \cdot {v_r}^2 \cdot \frac{\vec{v}_r}{v_r} \qquad \textbf{(1.44)}$$

$$\vec{\tau}_{Aero} = \sum_{i=1}^{n}\left(\vec{r}_{GC,i/CoM} \times \vec{F}_{Aero,i}\right) \quad \textbf{(1.45)}$$

Where *n* is the number of satellite surfaces which has the effective angle (0−90◦) with the velocity vector, $\vec{v}_r = \vec{v}_{ECI}\text{-}\vec{\omega}_E \times \vec{r}_{ECI}$ being $\vec{\omega}_E$ the Earth's rotation angular velocity, CD is the drag coefficient and it depends on the geometry and the size of satellite's faces, $A_{D,i}$ is the area of the i^{th} surface that it exposed to ram directions, ρ is the air density that depends on the altitude(**fig 1.13**) and $\vec{r}_{GC,i/CoM}$ is the vector from the center of mass to the area center of the i^{th} surface.

6.4. Residual Dipole Torque

In LEO, the interaction between the Earth's magnetic field and the magnetic dipole of a satellite generates a torque. Many satellites take advantage of this by using either electromagnets or permanent magnets for spacecraft attitude control. In addition to the designed magnetic dipole, all spacecraft have an unintended dipole, known as a residual dipole, that is caused by electric currents and magnetic material within the satellite [13].

The moment caused by the interaction between the residual dipole and Earth's magnetic field is known as the residual dipole moment. For small satellites this torque can be the main disturbance torque as well as the gravity gradient torque.

$$\vec{\tau}_{res} = \vec{m}_{res} \times \vec{B}_E \quad \textbf{(1.46)}$$

Where $\vec{m}_{res}$ is the magnetic moment of the residual dipole and $\vec{B}_E$ is the Earth's local magnetic field.

7. Actuators

The attitude control of a spacecraft can be considered being either actively controlled (meaning that a controller calculates necessary control torques and acting on the satellite to adjust its attitude to a desired position) or passively controlled (meaning that the satellite uses external torques that occurs due to its interaction with the environment and thus they cannot be avoided, in this case the disturbances being used for forcing the attitude of the satellite). In general,

the active control assures 3 axis stabilization, while the passive control gives the opportunity for 1 axis stabilization. Some disadvantages points of the passive actuators are that the pointing accuracy is pretty bad and also that the natural damping is very small meaning that additional energy dissipation devices need to be installed on board of the spacecraft.

7.1. Reaction Wheels

Reaction wheels are used as the primary attitude control actuators on most spacecraft. This actuator uses the rotational variant of Newton's third law. If the action is accelerating a wheel inside the spacecraft, the spacecraft will accelerate just as much in the opposite direction. In other words, reaction wheels are simple disks (rotors) that are spun by an electric motor. When the motor applies a torque to speed up or slow down the rotor, it produces a reacting torque on the body of the satellite,

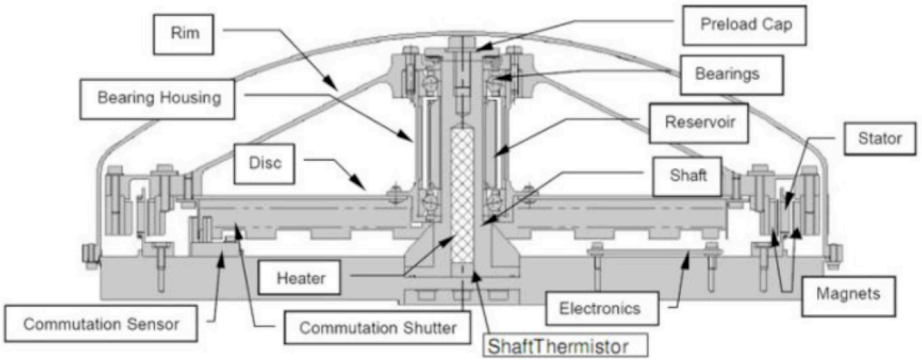

Fig 1.14) Reaction wheels design.

Reaction wheel failures have been a problem on many space missions. Providing extra reaction wheels for redundancy gives some protection, but failure of one reaction wheel is often followed by failure of other wheels of the same design on the same spacecraft. However, new configurations of reaction wheels as the tetrahedron, makes up the most accurate attitude control actuator for satellites. The size of today's systems is so large that this solution is not suited for CubeSats.

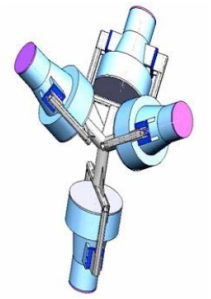

Fig 1.15) Reaction wheels in tetrahedron configuration.

7.2. Magnetic Torquers

Most of the time, magnetic torquers are used as coils (torque coil is simply a long copper wire, winded up into a coil) but more generally any conducting device can be used to perform this function. They are often used in combination with angular momentum storage and exchange devices, or they are used to unload momentum accumulated by the reaction wheels without using thrusters.

The physical principle consists in generating a magnetic dipole moment in a desired direction, moment which will interact with the Earth's magnetic field and thus external torques appear. However, the use of such a device is limited to the low Earth orbits where the Earth's magnetic field strength has usable values and should take in consideration that the generation of torques can be done just for the one perpendicular to the magnetic field vector.

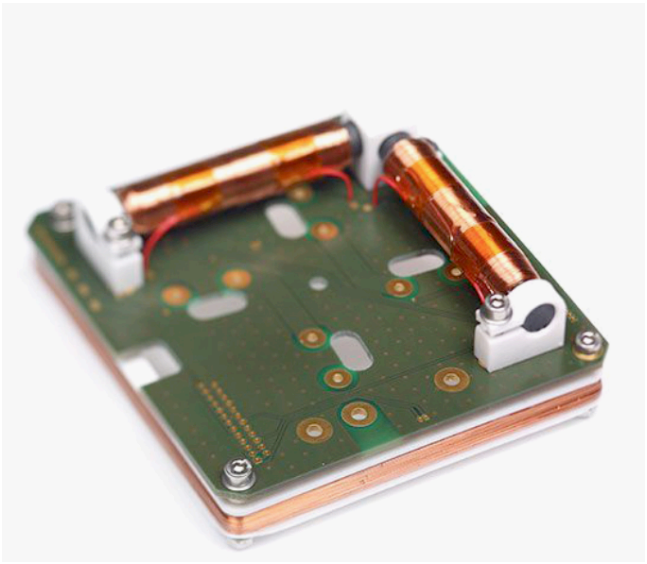

Fig 1.16) Magnetorquer.

7.3. Thrusters

The torques generated by the thrusters is considered as external torques since the angular momentum of the entire satellite changes. The accuracy of the attitude control depends on the minimum impulse of the type of thruster used. Thrusters generate both forces and torques, so they can be used for both trajectory control and attitude control.

Depending on the size of the satellite and taking in consideration the complexity of this solution, different types of thrusters are normally being used: gas jets, ion jets or even nuclear propulsion. Gas jets produce thrust by a collective acceleration of propellant molecules, with the energy coming from either a chemical reaction or thermodynamic expansion.

Gas jets are classified as hot gas when the energy is derived from a chemical reaction or cold gas when it is derived from the latent heat of a phase change, or from the work of compression if no phase change is involved. Hot gas jets generally produce a higher thrust level (>5 N) and a greater total impulse or time integral of the force. Cold gas systems operate more consistently, particularly when the system is operated in a pulsed mode, because there is no chemical reaction which must reach steady state. The lower thrust levels (<1N) of cold gas systems may facilitate more precise control than would be available with a high thrust system.

Ion jets accelerate individual ionized molecules electro-dynamically, with the energy ultimately coming from solar cells or self containing electric generators.

Ion jets are primarily used in applications that do not require large amounts of thrust, such as satellite control. The thrust produced by a typical ion engine is on the order of millinewtons, and thus cannot yet be used as a primary propulsion system for launching any spacecraft from the earth's surface. However, the spacecraft utilizing the ion jet engine can be launched via chemical rocket, and then sufficient thrust can be developed in the frictionless atmosphere of space.

$$\vec{F}_{T\hbar} = -\dot{m} \cdot v_{rel} \quad \textbf{(1.47)}$$

$$\vec{\tau}_{T\hbar} = \vec{r}_{CM,T\hbar} \times \vec{F}_{T\hbar} \quad \textbf{(1.48)}$$

Where $\dot{m}$ is the rate at which mass is expelled, v_{rel} is the velocity of the expelled mass relative to the spacecraft and $\vec{r}_{CM,Th}$ is the vector from the center of mass of the spacecraft to the thruster.

Fig 1.17) Cold gas thruster.

7.4. Gravity Gradient Boom

Gravity gradient is used in stabilization mode as a passive attitude control method. Basic requirement in applying gravity boom is that the gravity gradient torque must be greater than all other environmental torques.

They require no power from satellite and maintain stable orientation relative to central body such as Earth. But their control accuracy is limited.

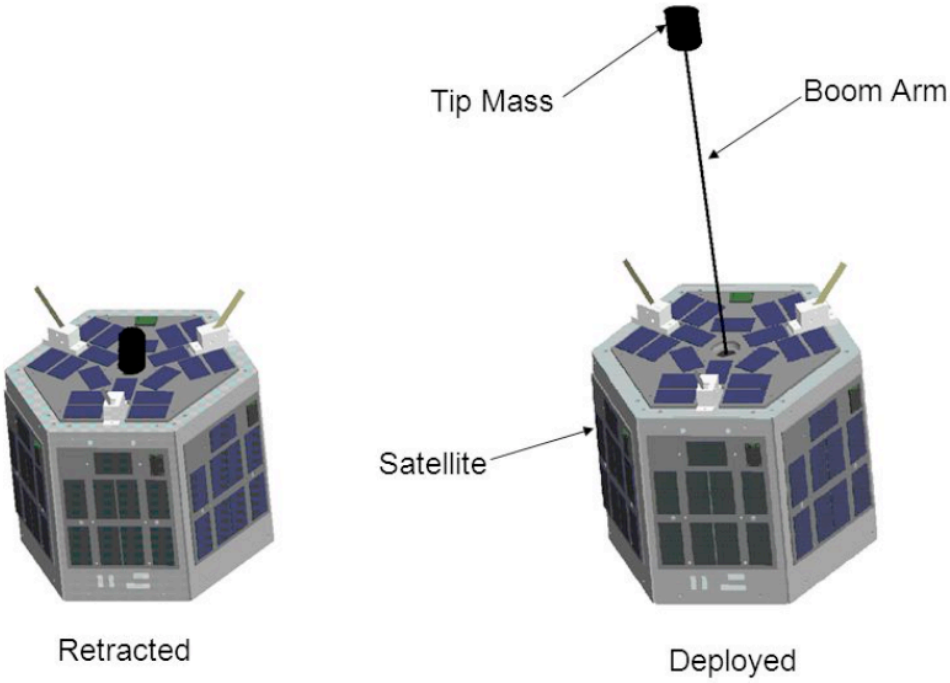

Fig 1.17) Gravity Gradient Stabilization.

CHAPTER 3: SIMULATION AND RESULTS

In this chapter we are going to implement all the theory seen before in a program done in Mat Lab.

It is important to verify if the program works properly following the laws of physics. For this, there will be a section called validations where simple performances of the CubeSat are going to be simulated to contrast with reality.

Finally, we are going to run our program with different parameters (orbital, initial conditions) to see how does our CubeSat behaves in front of the external torques and disturbances.

8. Software validation

Starting off from a simple situation that everyone can imagine is that CubeSat is orbiting around the Earth without the effect of external perturbations.

Initial conditions for the satellite (considering a homogeneous solid), are in the following table:

Features	Value
Initial angular velocity	{0, 0, 0} [rad/s]
Initial Euler angles	{0, 0, 0} [º]
Inertia Tensor	$I_{xx} = I_{yy} = I_{zz}$ =0.00221 [kg·m^2]
Weight	1,33 [kg]

Table 2.1. Case 1: Satellite at rest without effect of disturbances.

Orbital parameters for the simulation are listed below:

Features	Value
Orbit Altitude	650 [km]
Orbital Period	97,5 [min]
Orbital Inclination	0 [º]
Argument of Periapsis	97 [º]
Longitude Ascending Node	131 [º]
Eccentricity	0

Table 2.2. Orbital parameters.

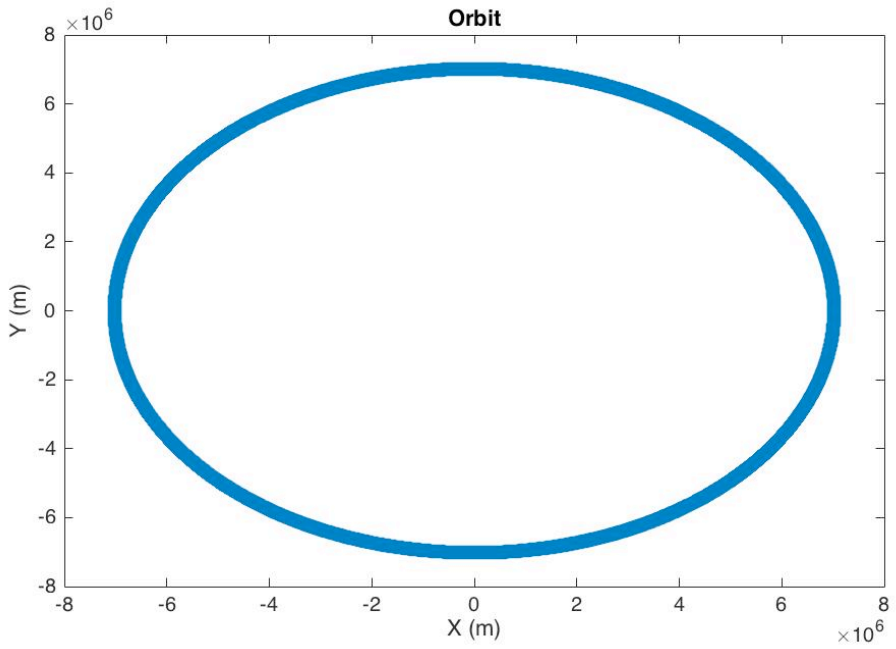

Fig 2.1) Orbit in X-Y plane.

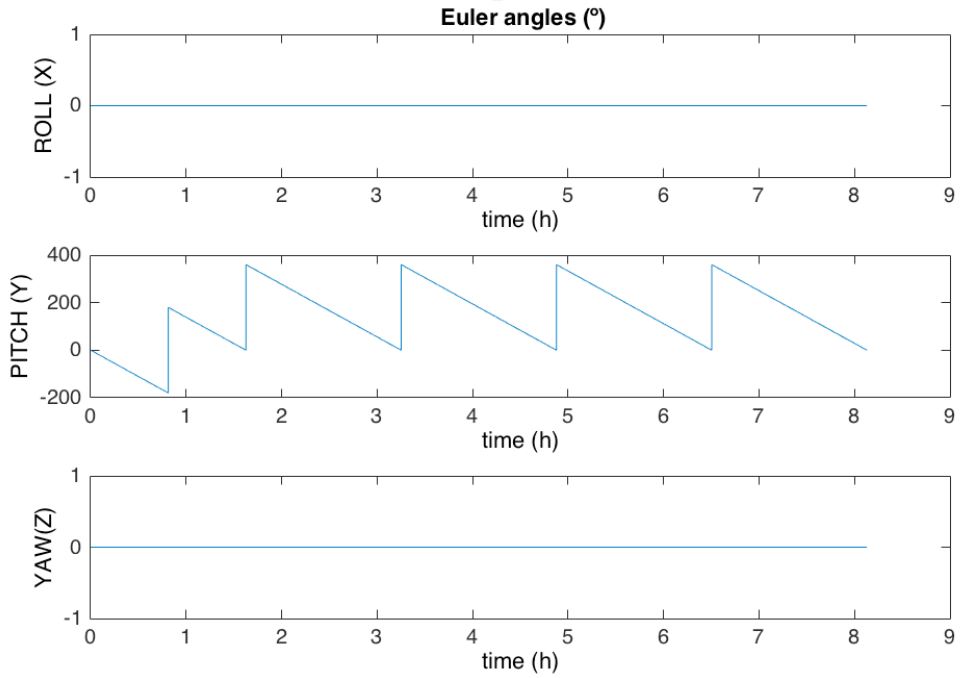

Fig 2.2) Satellite performance: Euler Angles (Case 1).

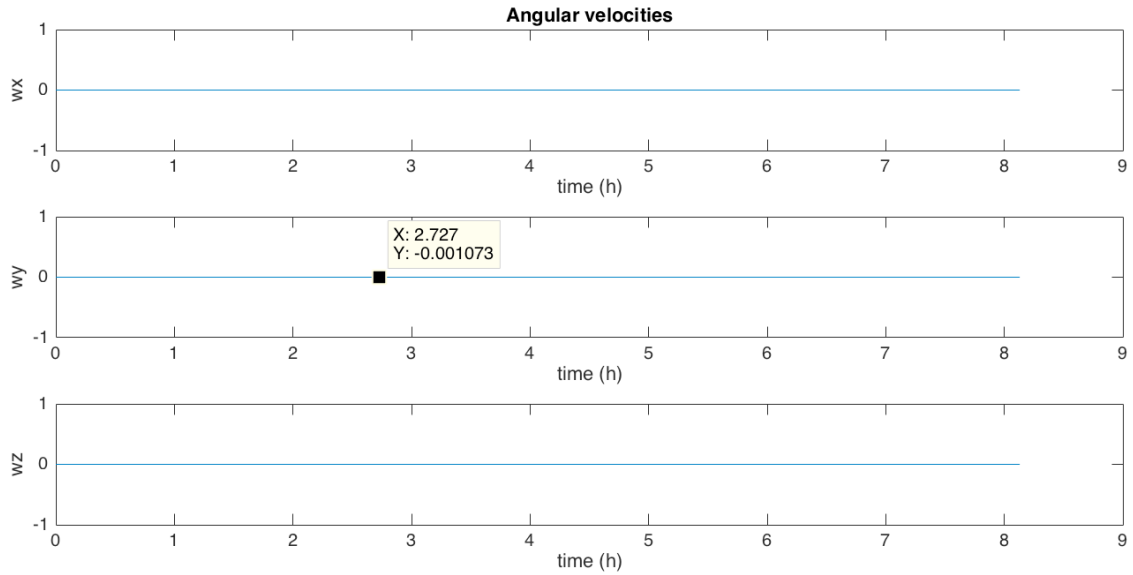

Fig 2.3) Satellite performance: Angular Velocity.

As mentioned before, this case is only the motion of the satellite around the Earth in the orbit (circular orbit) with no effect of disturbances. In **Fig 2.3** is depicted the angular velocity in the different axes of our 1U CubeSat. For axes X and Z, we have no angular velocity, it is normal because initial conditions are

null. In axis Y, we have an orbital angular velocity that depends on ECI position and velocity that makes turn the satellite and has a value of 0.00103 rad/s. The angle that changes during the turn of this angular velocity is the pitch angle. We can see in **Fig 2.2** that satellite will be continuously turning from 0º to 360º only by the effect of the orbit.

Now let's change the conditions, now initial Euler angles will not be zero as well as the initial angular velocities, this will simulate the deployment of the CubeSat with random values, but also without the effect the external moments and disturbances in the same orbit as before.

Features	Value
Initial angular velocity	{0.001, 0, 0.0002} [rad/s]
Initial Euler angles	{20, 40, 10} [º]
Inertia Tensor	$I_{xx} = I_{yy} = I_{zz}$ =0.00221 [kg·m^2]
Weight	1,33 [kg]

Table 2.3. Case 2: Random initial values for Euler angles and ang. velocities without effect of disturbances.

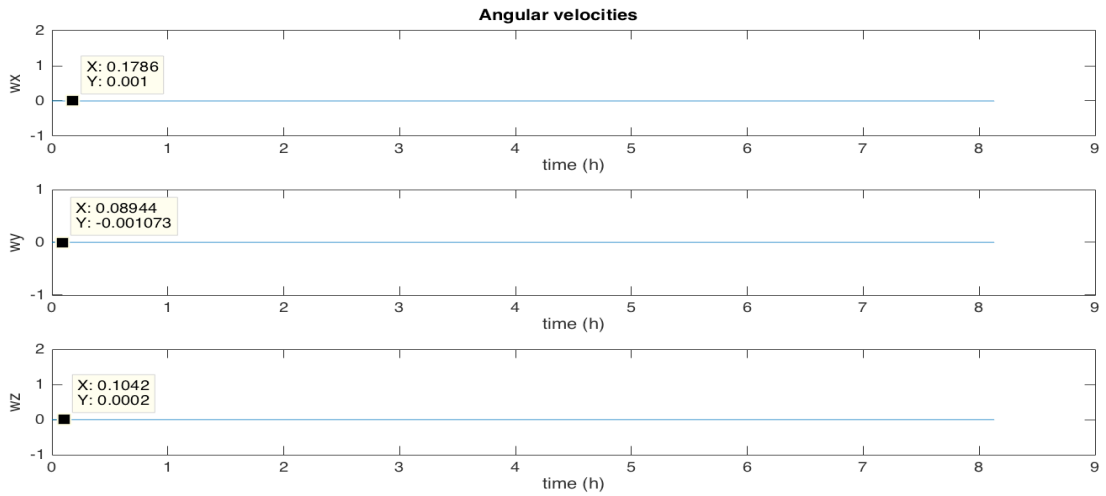

Fig 2.4) Satellite performance: Angular Velocity (Case 2).

In **Fig 2.4** we see that the angular velocity remains constant if no external forces are acting upon the satellite. Despite the initial angular velocity in y-axis is zero, we have the orbital angular velocity that makes it turn as in case 1.

As we have constant angular velocity in the three axes, we see that the motion is periodic for pitch that starts from 40 degrees and comes oscillatory from -15 degrees to 80 degrees. Also yaw has an oscillatory motion from -109 degrees to 11.5 degrees. For roll we have that it will be turning along the orbit starting from 20 degrees. We can see the representation below in **Fig 2.5**.

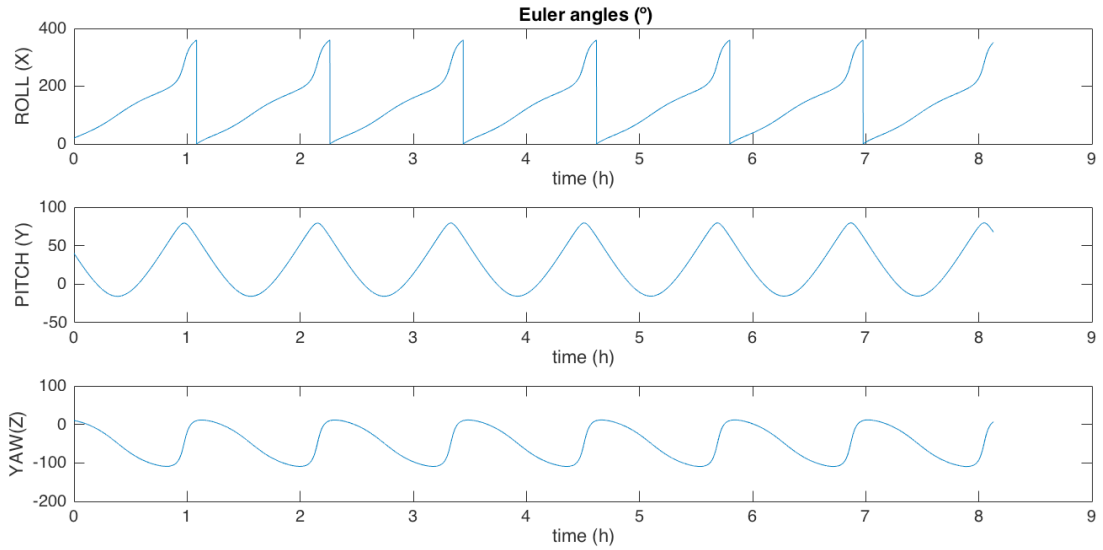

Fig 2.5) Satellite performance: Euler Angles (Case 2).

To conclude this section of software validation, we are going to add a torque but in a certain time it will disappear, so it means that angular velocity that torque was acting on, will increase linearly but when this moment stops, angular velocity will be constant.

Features	Value
Initial angular velocity	{0, 0, 0} [rad/s]
Initial Euler angles	{0, 0, 0} [º]
Torque	{0 , $3 \cdot 10^{-7}$, 0} [N·m]

Table 2.4. Case 3: Initial values with a moment acting for a certain time.

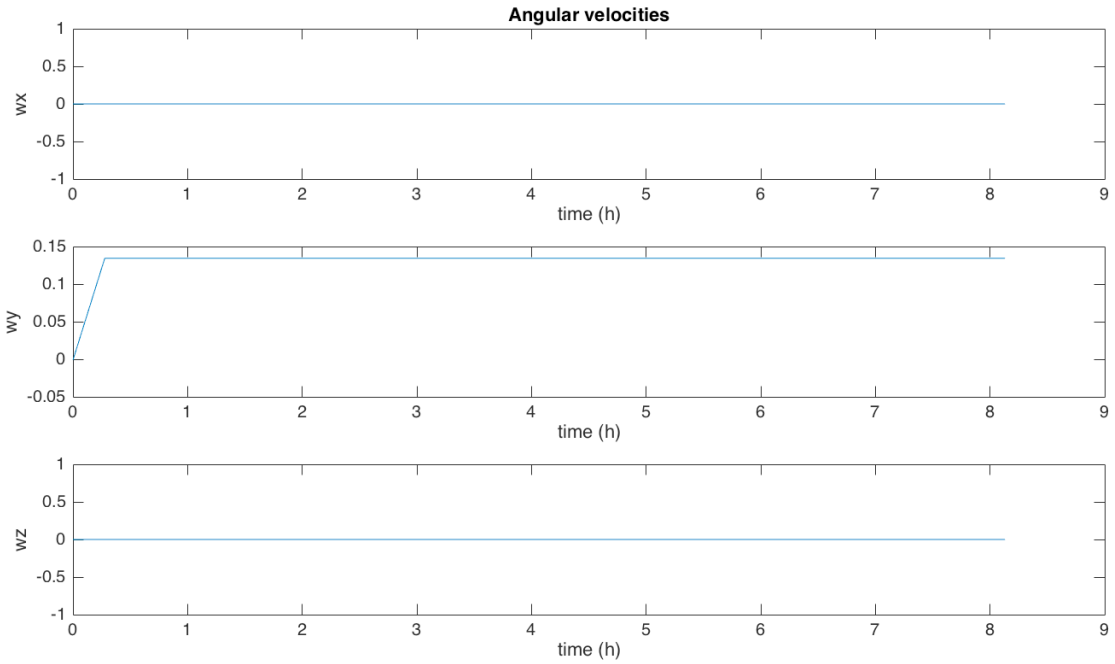

Fig 2.6) Satellite performance: Angular Velocity (Case 3).

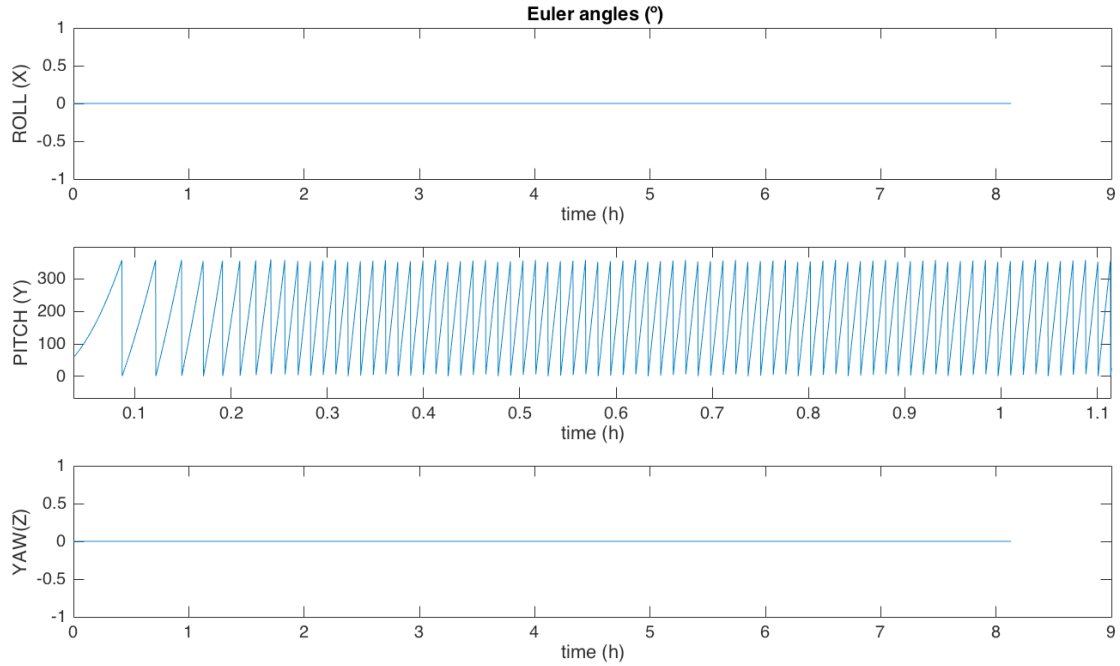

Fig 2.7) Satellite performance: Euler Angles (Case 3).

For this third case, we have included a torque for 1000 seconds, in **Fig 2.6** for y-axis angular velocity we see that velocity increases until torque stops and then it remains constant as we have expected. For Euler angles we see that only pitch changes during time periodically in **Fig 2.7** shows the turning of the

satellite. Also we have to recall that in y-axis orbital angular velocity is causing the turning of it as well.

9. Simulation results with disturbances and external torques

In this section is going to be simulated real cases where external torques and disturbances are acting on the satellite. Initial conditions, features of the orbit environment and satellite are listed in the tables.

Time simulations are **5 orbits** around the Earth, it varies depending on altitude.

Features	Value
Initial angular velocity	{0, 0, 0} [rad/s]
Initial Euler angles	{0, 0, 0} [º]
Orbit Eccentricity	0
Orbit Inclination	0 [º]
Drag Coefficient (CD)	2
Distance between the geographical center and the center of mass	{0.001, 0, -0.009} [m]
Reflectivity Effect Factor	0.21
Residual Dipole Vector	{ 0, 0.005, 0} [$A \cdot m^2$]
Density	$2{,}27 \cdot 10^{-12}$ [kg/m^3]
Inertia Tensor	$I_{xx} = I_{yy} = I_{zz} = 0.00221$ [$kg \cdot m^2$]
Weight	1,33 [kg]

Table 2.5. Environment of the orbit parameters, satellite and initial conditions.

For the first case, we will place a 1U CubeSat in an equatorial orbit at 650 km of altitude.

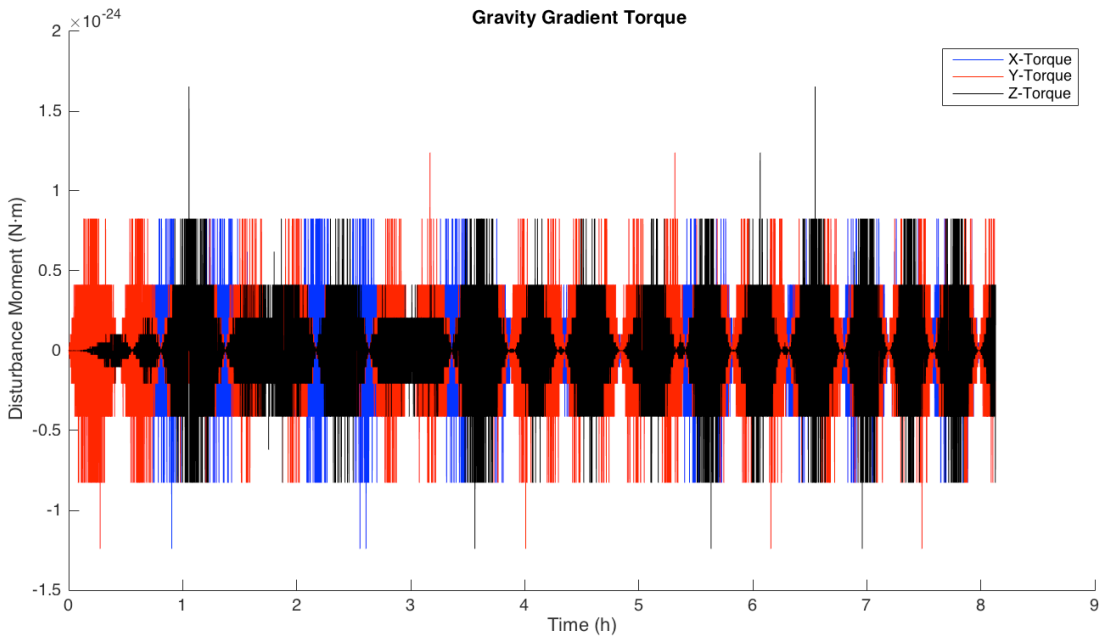

Fig 2.8) Gravity Gradient Torque (Case 1).

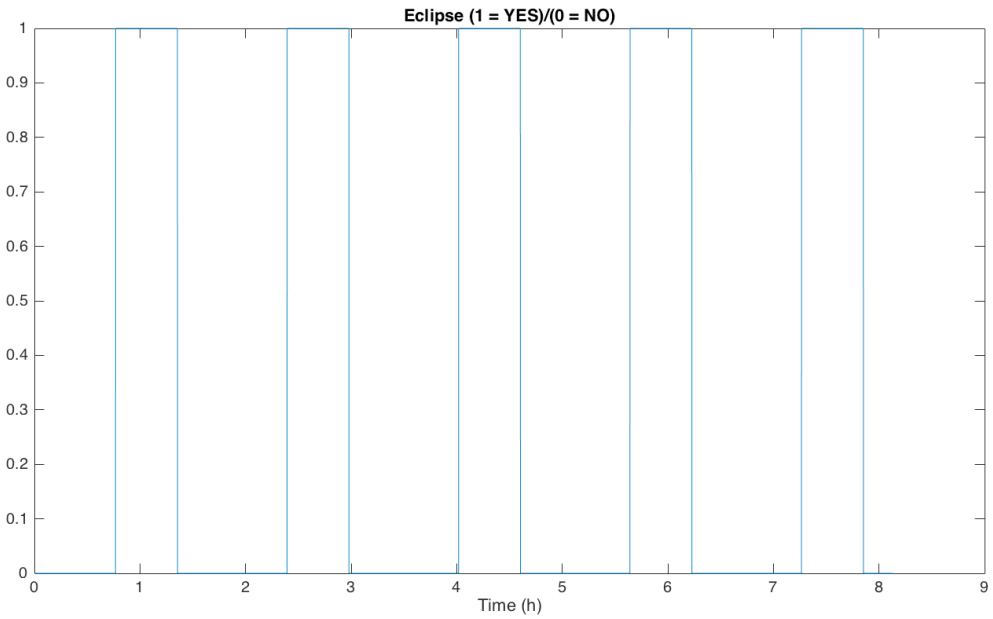

Fig 2.9) Eclipse Model.

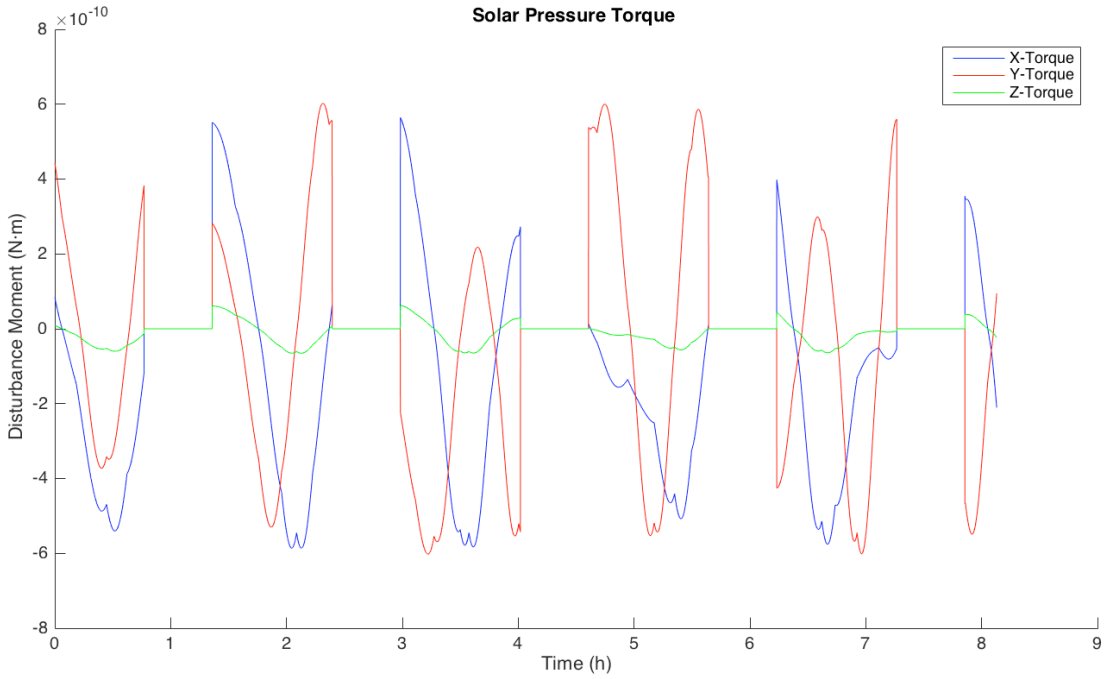

Fig 2.10) Solar Pressure Radiation Torque (Case 1).

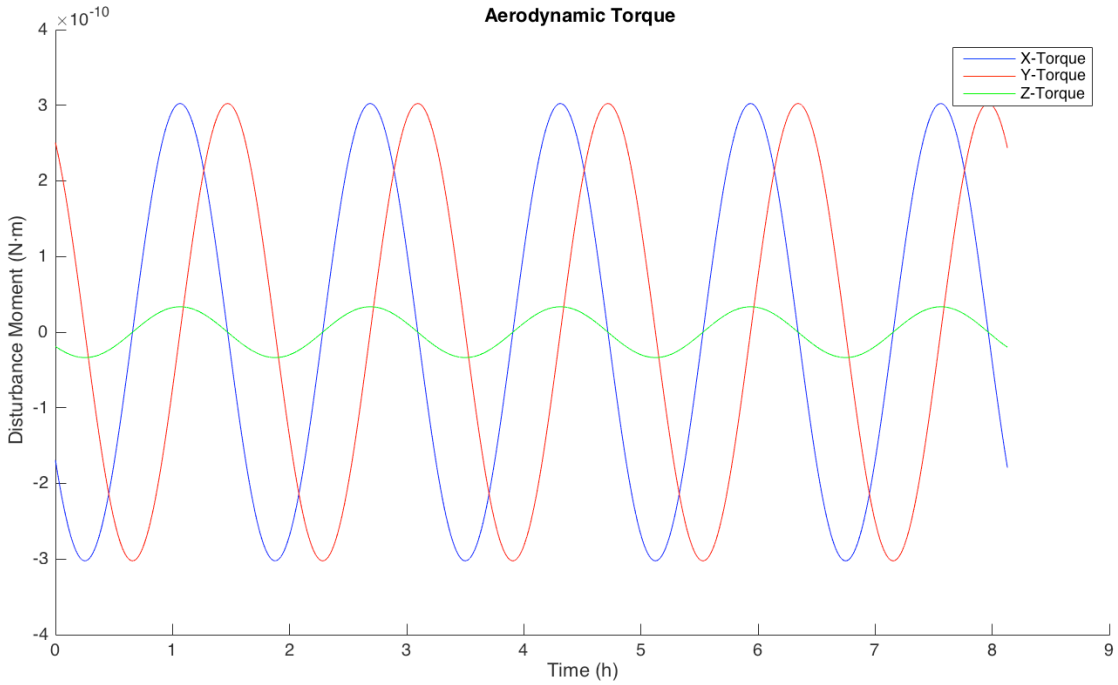

Fig 2.11) Aerodynamic Torque (Case 1).

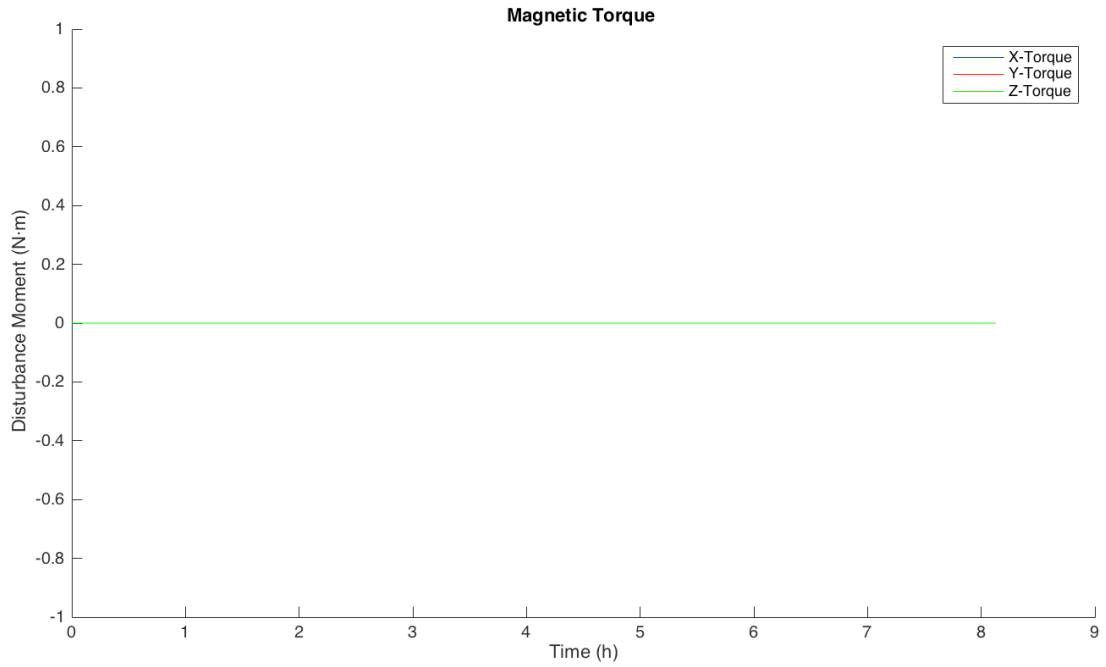

Fig 2.12) Magnetic Torque (Case 1).

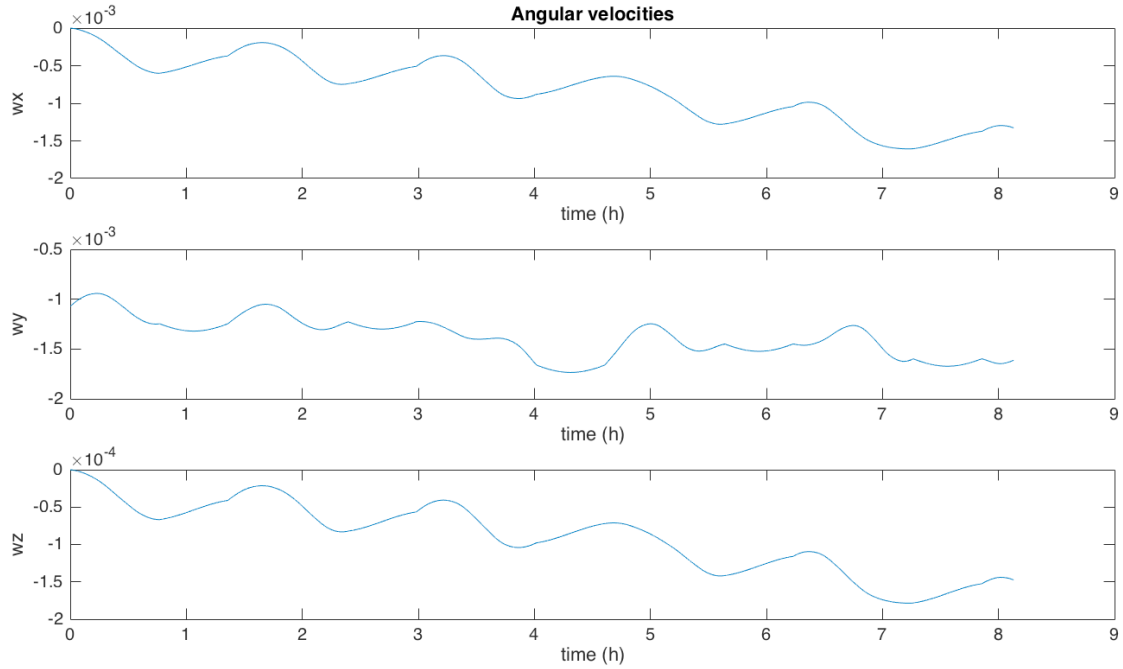

Fig 2.13) Angular Velocities (Case 1).

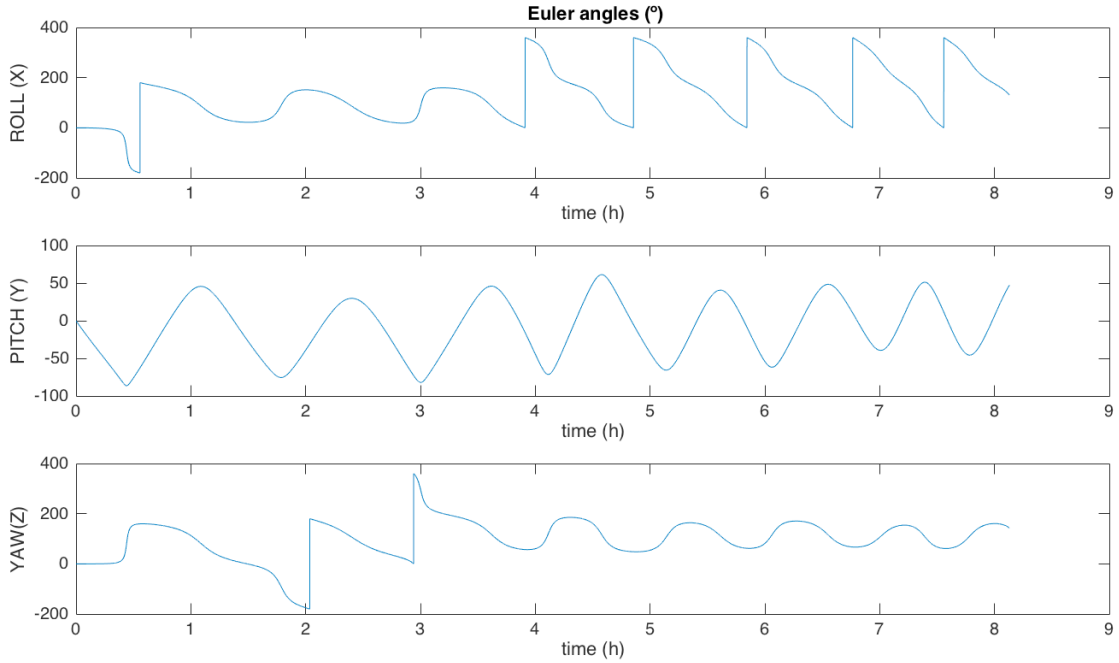

Fig 2.14) Euler Angles (Case 1).

Eclipse model shown in **Fig 2.9,** varies horizontally depending on the orbital velocity and the shape of the orbit, because is not the same the time of the satellite in umbra for an elliptical orbit than a circular orbit. Also as we stated that simulation times were going to be five orbits, we will have 5 times that satellite passes umbra region.

From **Fig 2.12** we can see that magnetic torque is null when the orbit is in the equatorial plane (orbital inclination is zero).

Related to angles of Euler in **Fig 2.14**, we see that becomes periodic when motion has reached 4h of the simulation. This makes possible to add a control without any problems.

Features	Value
Initial angular velocity	{0, 0, 0} [rad/s]
Initial Euler angles	{0, 0, 0} [º]

Orbit Eccentricity	0
Orbit Inclination	10 [º]
Drag Coefficient (CD)	2
Distance between the geographical center and the center of mass	{0.001, 0, -0.009} [m]
Reflectivity Effect Factor	0.21
Residual Dipole Vector	{ 0, 0.005, 0} [$A \cdot m^2$]
Density	$2{,}27 \cdot 10^{-12}$ [kg/m^3]
Inertia Tensor	$I_{xx} = I_{yy} = I_{zz} = 0.00221$ [$kg \cdot m^2$]
Weight	1,33 [kg]

Table 2.6. Parameters: Case 2.

For the second case, we will place a 1U CubeSat in the same orbit as in the previous section (in a circular orbit with an altitude of 650 km) but with an orbit inclination of 10 º, we will see how magnetic field affects satellite's attitude in a huge way.

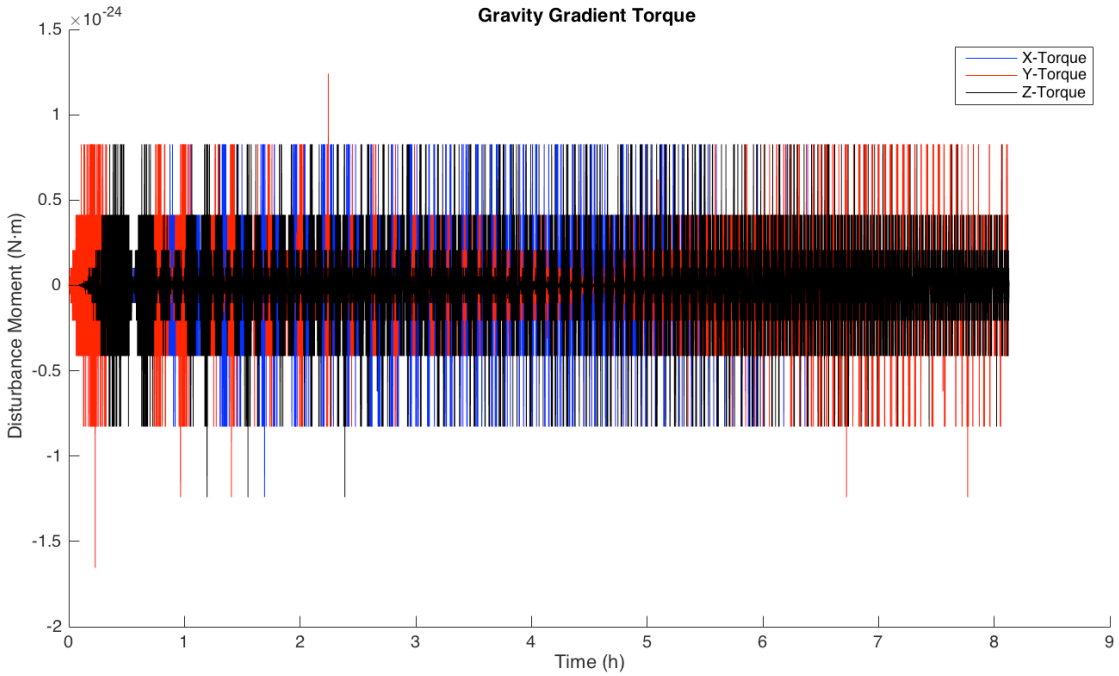

Fig 2.15) Gravity Gradient Torque (Case 2).

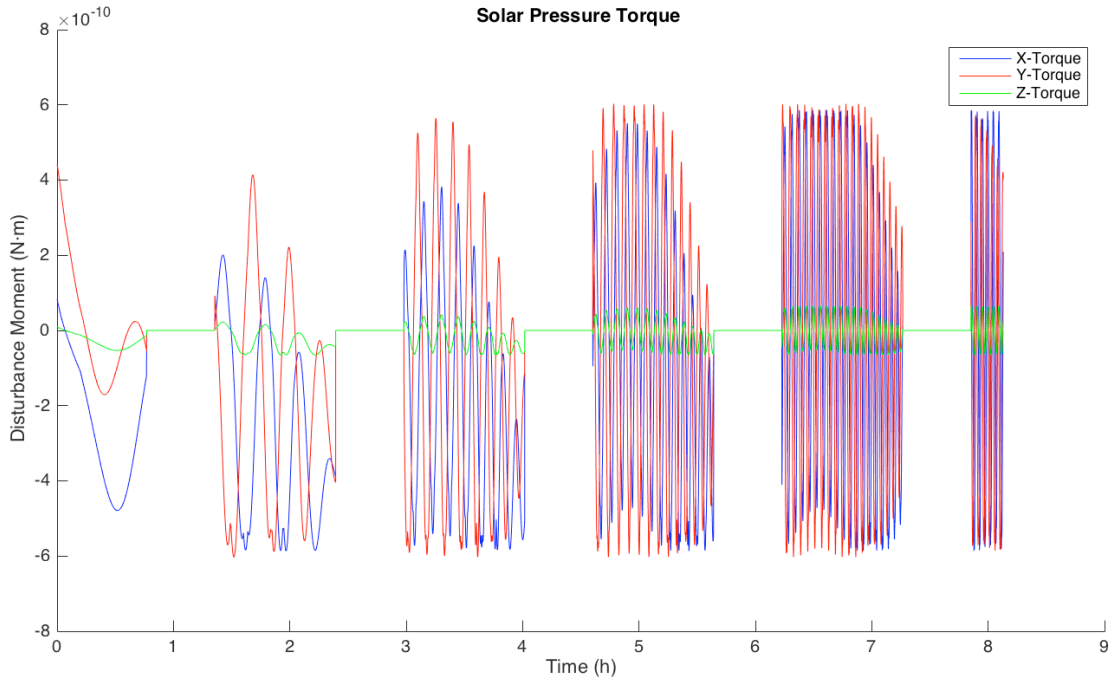

Fig 2.16) Solar Pressure Radiation Torque (Case 2).

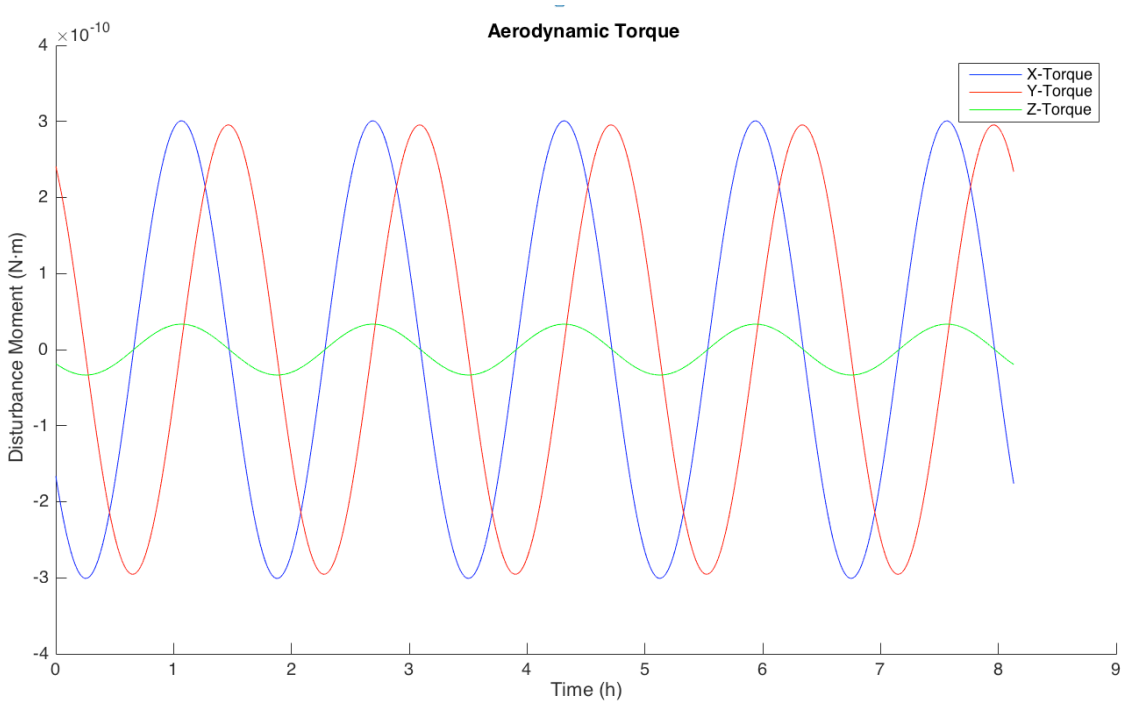

Fig 2.17) Aerodynamic Torque (Case 2).

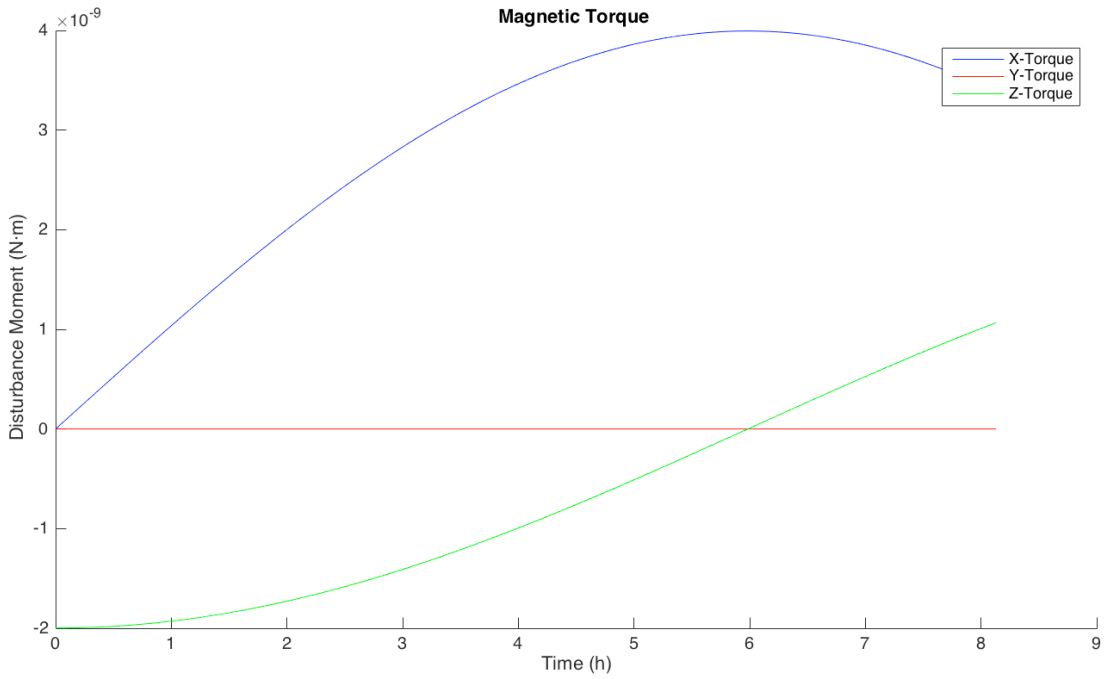

Fig 2.18) Magnetic Torque (Case 2).

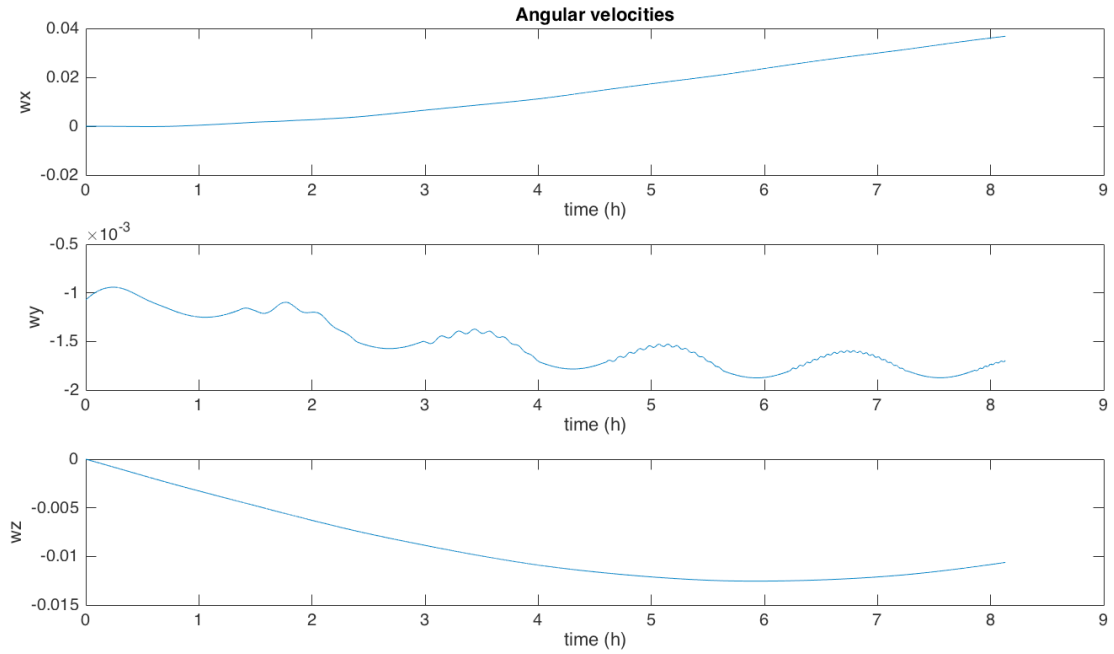

Fig 2.19) Angular Velocities (Case 2).

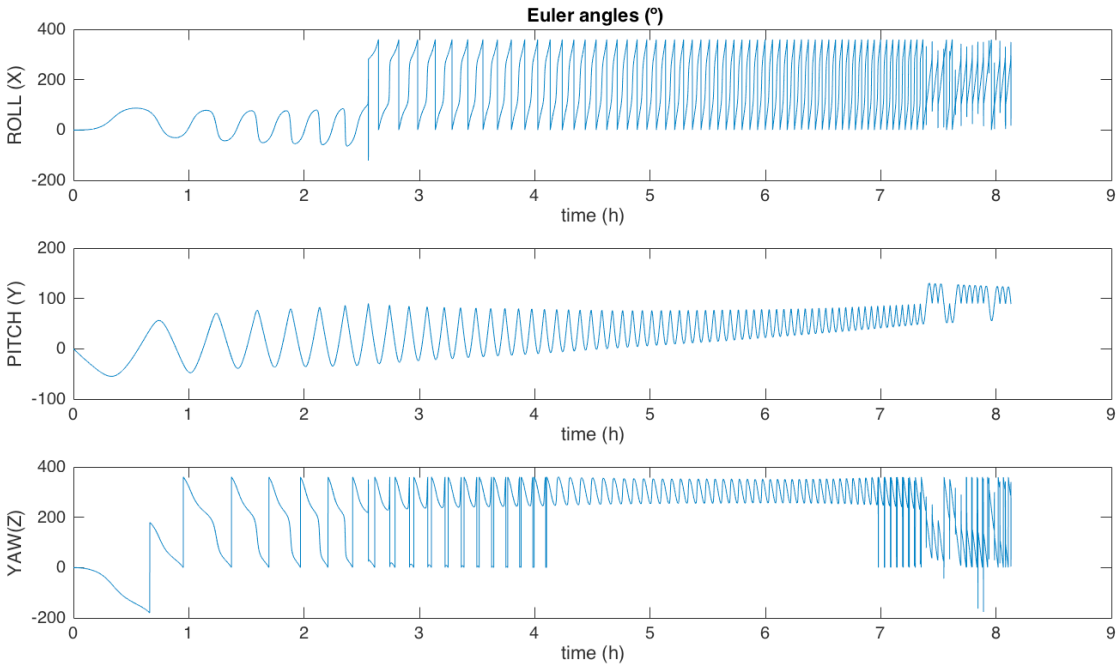

Fig 2.20) Euler Angles (Case 2).

Now magnetic field is acting upon the satellite, making it to increase the angular velocity in axes x and z while in y-axis angular velocity has the same order as in first case (see **Fig 2.19** and **Fig 2.13**).

As we increase the angular velocity, the change in Euler angles will be more abrupt for a short time than in first case that angular velocity is not fast enough (see **Fig 2.20** and **2.14**). Also in **Fig 2.16** we see how impacts the high angular rate in solar pressure radiation torque, if CubeSat turns quickly we will have more exposition of solar radiation on the walls of the satellite, that's why over time and while angular velocity increases in three axes the graph becomes more colourful.

Features	Value
Initial angular velocity	{0, 0, 0} [rad/s]
Initial Euler angles	{0, 0, 0} [º]
Orbit Eccentricity	0.5
Orbit Inclination	20 [º]
Drag Coefficient (CD)	2
Distance between the geographical center and the center of mass	{0.001, 0, -0.009} [m]
Reflectivity Effect Factor	0.21
Residual Dipole Vector	{ 0, 0.005, 0} [$A \cdot m^2$]
Density	$2{,}27 \cdot 10^{-12}$ [kg/m^3]
Inertia Tensor	$I_{xx} = I_{yy} = I_{zz} = 0.00221$ [$kg \cdot m^2$]
Weight	1,33 [kg]

Table 2.7. Parameters: Case 3.

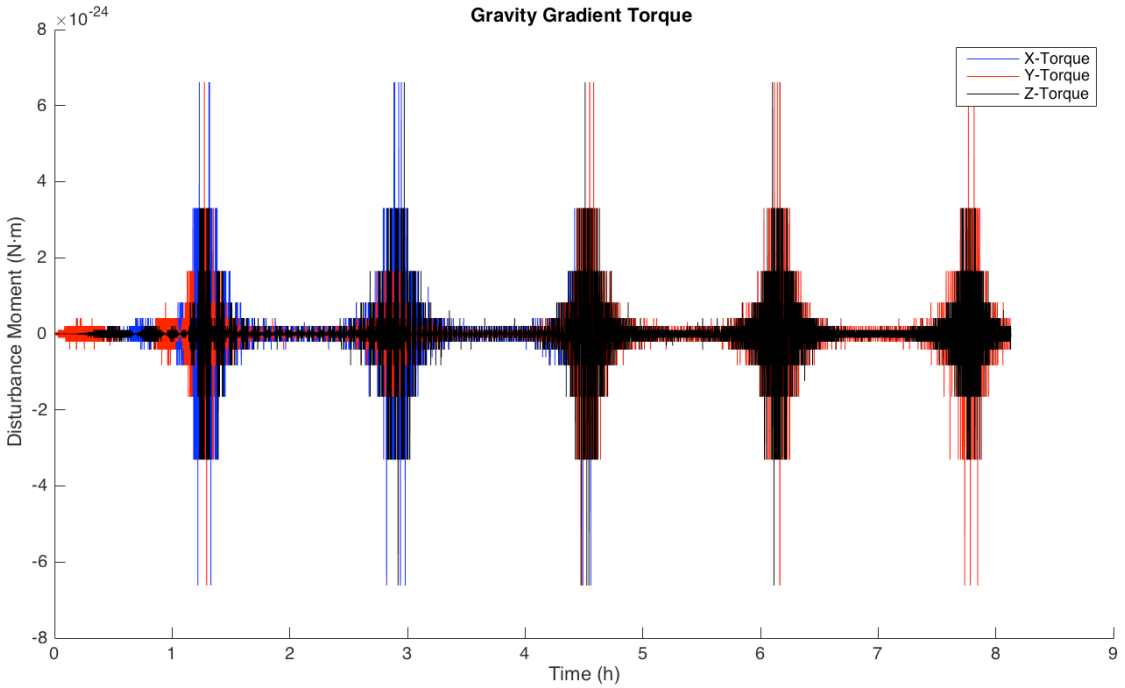

Fig 2.21) Gravity Gradient Torque (Case 3).

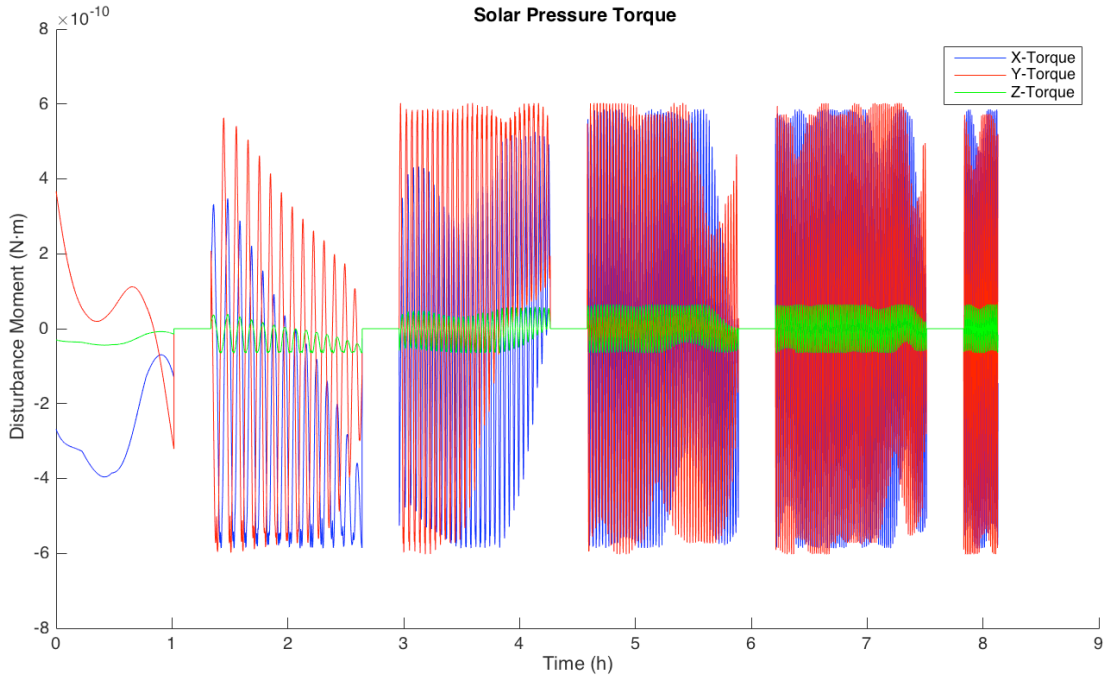

Fig 2.22) Solar Pressure Radiation Torque (Case 3).

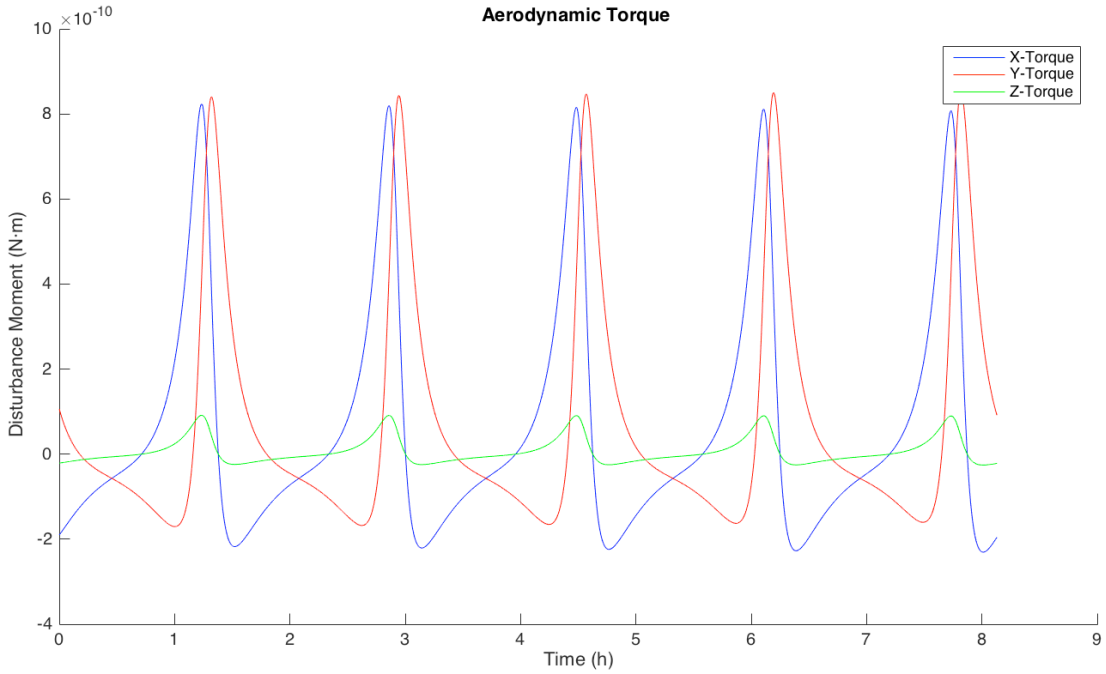

Fig 2.23) Aerodynamic Torque (Case 3).

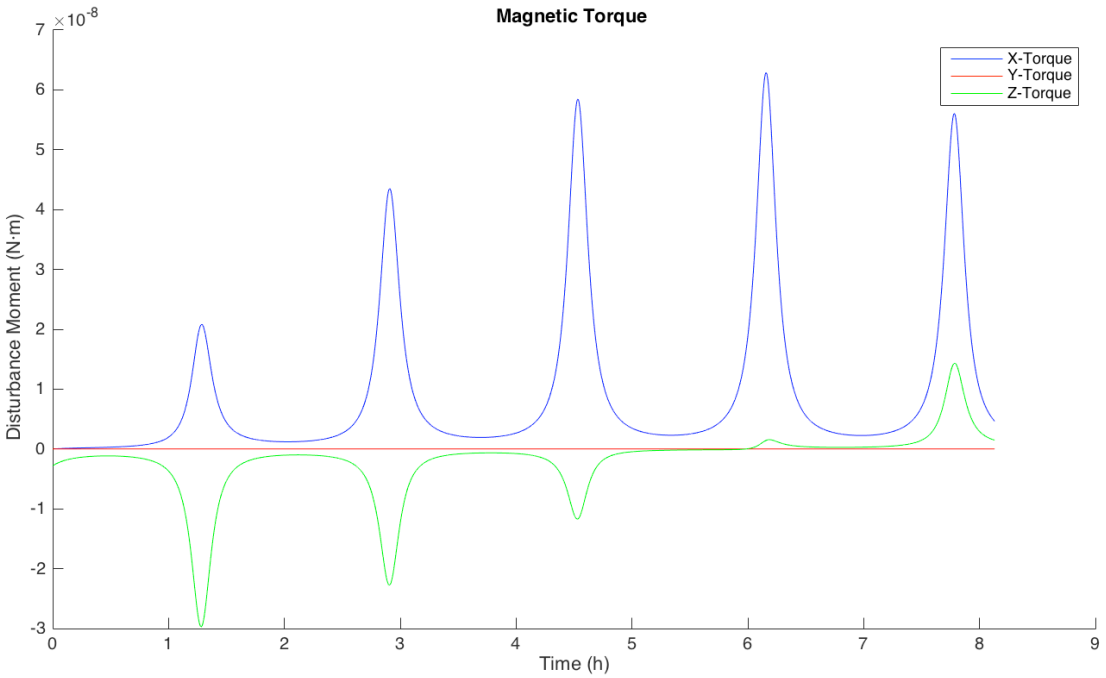

Fig 2.24) Magnetic Torque (Case 3).

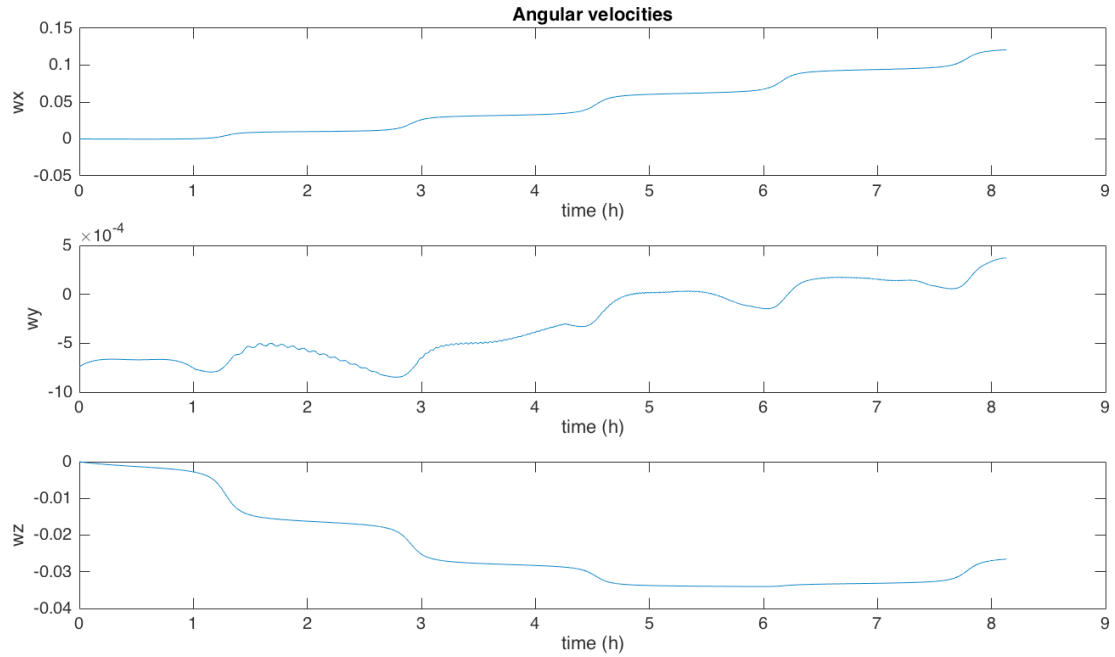

Fig 2.25) Angular Velocities (Case 3).

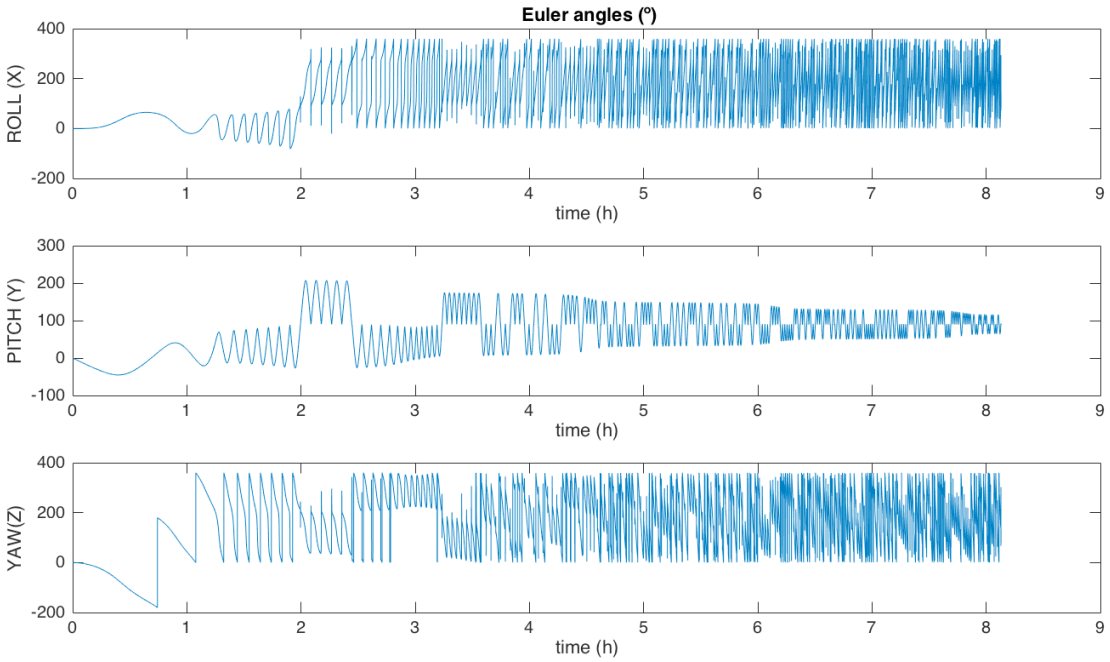

Fig 2.26) Euler Angles (Case 3).

CHAPTER 4: CONCLUSIONS

In this project we have achieved an accurate simulator of attitude representation on LEO orbits by introducing only the orbital parameters and the size of the satellite as well as its tensor of inertia. Even though we have focused on 1U CubeSats this simulator can represent the attitude of all kind of satellite shapes while the inertia tensor is included as input in our simulator.

We have used Matlab as a programming tool, for this reason, simulation times have been short because of the time that spends the program in order to get the results. This project mainly should have been done using C++ software because it is faster than Matlab and we could have worked with large simulation times, but Matlab was handier for mathematical computations.

In all simulations, the external moment that has more impact in Low Earth Orbits is the torque created by the Earth's magnetic field. Another moment that is dominant on LEO is the torque generated by the gravity gradient, but as we did not use any kind of passive control such as a gravity boom that makes the z-moment of the gravity gradient null, but affects highly to the other two axes (x and y) over an order of $1 \cdot 10^{-9}$ N·m, we have for our 1U CubeSat a low moment created on the three axes of the order of $1 \cdot 10^{-24}$ N·m.

Generally, an incorrect design, for instance a bad distribution of mass in the satellite initially rotations can be stable but over time become in unstable rotations. This kind of performances must be avoid, we want periodic and stable rotations to use control.

As expected, we can see that in all cases from Section 9, over time the turns of the satellite become stable and periodic, that enables to add a controller. The most common active controllers for CubeSats are reaction wheels because are suitable for small satellites.

Control must be used along the satellite's life in order to get our satellite pointing in the direction is desired.

CHAPTER 5: REFERENCES

[1]- https://sma.nasa.gov/docs/default-source/News-Documents/cubesat-data-analysis.pdf?sfvrsn=b106e4f8_0 (Estudio de numero de CubeSats lanzados - introducción).

[2]- https://www.esa.int/Education/CubeSats_and_Education_the_Fly_Your_Satellite!_programme

(ESA PROGRAMME - fly your satellite Programme)

[3]- https://brownspace.org/acds/ (Active and passive control system)

[4]- Paluszek M., Razin Y. , Pajer G.,Mueller J. and Thomas S., " *Spacecraft Attitude and Orbit Control* Vol. 1: A Systems Approach, Third Edition", Princeton Satellite System. Inc.

[5]- https://www.grc.nasa.gov/www/K-12/airplane/newton.html. (Definition of Newton Laws)

[6]- «Celestrak,» 16 December 2000. [Çevrimiçi]. Available: https://www.celestrak.com/columns/v02n01/.

[7]- F. Landis Markley, John L. Crassidis, " *Fundamentals of Spacecraft Attitude Determination and Control*", Springer, New York, 2014.

[8]- J. Wertz, *Spacecraft attitude determination and control*, Springer Netherlands, 1994.

[9]- Olmos, J., " *Ground Track of GPS satellites*", EETAC ,UPC, 2016.

[10]- Vázquez Valenzuela, R. , *"Teoría de Perturbaciones. Propagadores.",* Universidad de Sevilla, 2016.

[11]- Bona, B., Canuto, E., " *Dynamic Model of the Spacecraft Position and Attitude*", Dipartimento di Automatica e Informatica, Politecnico di Torino, Torino, Italy, 2002.

[12]- Hall, C., " Attitude Kinematics", Aerospace and Ocean Engineering, Virginia Tech.

[13]- Daeyoung Lee, John C. Springmann, Sara C. Spangelo and James W. Cutler , " *Satellite Dynamics Simulator Development Using Lie Group Variational Integrator*", Aerospace Engineering, University of Michigan, Ann Arbor, Michigan, USA,2011.

[14]- Svartveitk, K., Attitude determination of the NCUBE satellite, Master's thesis, Norwegian University of Science and Technology, Trondheim, Norway, Jan. 2003.

[15]- Dipole approximations of the geomagnetic field. Web page : https://www.spenvis.oma.be/help/background/magfield/cd.html

[16]- Jens Gießelmann , "*Development of an Active Magnetic Attitude Determination and Control System for Picosatellites on highly inclined circular Low Earth Orbits",*RMIT University, 2006.

CHAPTER 6: ANNEX

Functions:

```
%% Function Orbit Plane reference frame to Local Orbit
function [R_O_OP] = OrbitP_To_LocalOrbit(true_anom)


R_1_1 = -sin(true_anom);
R_1_2 = 0;
R_1_3 = cos(true_anom);

R_2_1 = cos(true_anom);
R_2_2 = 0;
R_2_3 = sin(true_anom);

R_3_1 = 0;
R_3_2 = 1;
R_3_3 = 0;


R_O_OP =  [R_1_1 R_1_2 R_1_3;
        R_2_1 R_2_2 R_2_3;
        R_3_1 R_3_2 R_3_3];

end




%% Function - Orbit to Body reference frame

function[R_B_O] = OrbitP_To_Body(roll,pitch,yaw)

% R_1_1 = cos(pitch)*cos(yaw);
% R_1_2 = cos(pitch)*sin(yaw);
% R_1_3 = sin(pitch);
%
% R_2_1 = cos(yaw)*sin(pitch)*sin(roll) - cos(roll)*sin(yaw);
% R_2_2 = cos(roll)*cos(yaw) + sin(pitch)*sin(roll)*sin(yaw);
% R_2_3 = cos(pitch)*sin(roll);
%
% R_3_1 = sin(roll)*sin(yaw) + cos(roll)*cos(yaw)*sin(pitch);
% R_3_2 = cos(roll)*sin(pitch)*sin(yaw) - cos(yaw)*sin(roll);
% R_3_3 = cos(pitch)*cos(roll);


R_1_1 = cos(pitch)*cos(yaw);
R_1_2 = cos(yaw)*sin(pitch)*sin(roll) - sin(yaw)*cos(roll);
R_1_3 = cos(yaw)*sin(pitch)*cos(roll) + sin(yaw)*sin(roll);

R_2_1 = sin(yaw)*cos(pitch);
R_2_2 = sin(yaw)*sin(pitch)*sin(roll) + cos(yaw)*cos(roll);
```

```
R_2_3 = sin(yaw)*sin(pitch)*cos(roll) - cos(yaw)*sin(roll);

R_3_1 = -sin(pitch);
R_3_2 = cos(pitch)*sin(roll);
R_3_3 = cos(pitch)*cos(roll);

R_B_O = [R_1_1 R_1_2 R_1_3;
        R_2_1 R_2_2 R_2_3;
        R_3_1 R_3_2 R_3_3];

end


%% Kepler To ECI

function [X,Y,Z,X_dot,Y_dot,Z_dot] = Kepler_To_ECI(w,i,Omega,r,t_anom,a_mom,p,e)

%Compute position in ECI coord

X = r*( cos(Omega)*cos( w + t_anom) - sin(Omega)*sin( w + t_anom)*cos(i));

Y = r*( sin(Omega)*cos( w + t_anom) + cos(Omega)*sin( w + t_anom)*cos(i));

Z = r*( sin(i)*sin( w + t_anom));

%Compute velocity in ECI coord

X_dot = ((X*a_mom*e)/(r*p))*sin(t_anom) - (a_mom/r)*(cos(Omega)*sin( w + t_anom)
+ sin(Omega)*cos( w + t_anom)*cos(i));

Y_dot = ((Y*a_mom*e)/(r*p))*sin(t_anom) - (a_mom/r)*(sin(Omega)*sin( w + t_anom) -
cos(Omega)*cos( w + t_anom)*cos(i));

Z_dot = ((Z*a_mom*e)/(r*p))*sin(t_anom) + (a_mom/r)*cos( w + t_anom)*sin(i);

end


%% Function ECI to Orbit reference frame

%https://adcsforbeginners.wordpress.com/2015/08/26/5-the-rotation-matrices/

%http://www.ladispe.polito.it/corsi/meccatronica/02JHCOR/2011-
12/Slides/Dynamics_revised.pdf
%(second matrix)

function [R_OP_ECI] = ECI_To_OrbitPlane(i_orbit,Omega)

% R_1_1 = -sin(w_peri + true_anom)*cos(Omega) - cos(w_peri +
true_anom)*cos(i_orbit)*sin(Omega);
% R_1_2 = -sin(w_peri + true_anom)*sin(Omega) + cos(w_peri +
true_anom)*cos(i_orbit)*cos(Omega);
% R_1_3 =  cos(w_peri + true_anom)*sin(i_orbit);
```

```
%
% R_2_1 = -sin(Omega)*sin(i_orbit);
% R_2_2 =  sin(Omega)*cos(i_orbit);
% R_2_3 = -cos(i_orbit);
%
% R_3_1 = -cos(w_peri + true_anom)*cos(Omega) + sin(w_peri +
true_anom)*cos(i_orbit)*sin(Omega);
% R_3_2 = -cos(w_peri + true_anom)*sin(Omega) - sin(w_peri +
true_anom)*cos(i_orbit)*cos(Omega);
% R_3_3 = -sin(w_peri + true_anom)*sin(i_orbit);

R_1_1 =  cos(Omega);
R_1_2 = -sin(Omega)*cos(i_orbit);
R_1_3 =  sin(Omega)*sin(i_orbit);

R_2_1 = sin(Omega);
R_2_2 = cos(Omega)*cos(i_orbit);
R_2_3 =-cos(Omega)*sin(i_orbit);

R_3_1 = 0;
R_3_2 = sin(i_orbit);
R_3_3 = cos(i_orbit);


R_OP_ECI = [R_1_1 R_1_2 R_1_3;
        R_2_1 R_2_2 R_2_3;
        R_3_1 R_3_2 R_3_3];



end

%% Kepler Propagator

% Kepler parameters:
%             a -> semimajor axis
%             e -> eccentricity
%             i -> orbit inclination
%             Omega -> longitude ascending node
%             w -> argument of periapsis
%             Mo -> Mean anomaly
%             inc_t -> time of aplication (increment)

function[true_anom,ang_mom,M_k,Ek,r_kepler] = Kep_propagator (a,e,Mo,inc_t)


%% Constants:

G=6.67384E-11; %gravity constant

M=5.972E24;    % Earth mass. (kg)

mu = G*M;

%% Formulas:
```

```
n=sqrt((mu)/a^3);% Angular speed


M_k= Mo + n*inc_t;% Mean anomaly

Ek=M_k;

D=1;
while(D>=1E-8) % We need to compute the value with a certain error. The loop stops
when we reach it
  Ek1=Ek;
  Ek=M_k+e*sin(Ek1);
  D=abs(Ek-Ek1);
end
% For Ek (Eccentric Anomaly) we take the last value of the whole iteration

%for true anomaly:
sinvk=(sqrt(1-e^2)*sin(Ek))/(1-(e*cos(Ek)));
cosvk=(cos(Ek)-e)/(1-e*cos(Ek));

true_anom=angle(complex(cosvk,sinvk));

% aa = sqrt((1 + e)/(1 - e));
% bb = tan(Ek/2);
% true_anom = 2*atan(aa*bb);
% true_anom = true_anom*180/pi;

r_kepler = a*(1 - e*cos(Ek)); % Radius r.

ang_mom = sqrt(mu*a*(1 - e^2));
end




%% Propagador J2 medio

% Formulas extraidas de la presentacion de teoria de perturbaciones de la
% universidad de sevilla (apartado: propagadores)

% Kepler parameters:
%           a -> semimajor axis
%           e -> eccentricity
%           i -> orbit inclination
%           Omega -> longitude ascending node
%           w -> argument of periapsis
%           Mo -> Mean anomaly
%           inc_t -> time of aplication (increment)
%           J_2 -> Harmonic coefficient
```

```
function[true_anom,ang_mom,M_k,Ek,r_kepler,w_k,Omega_k] = J2_propagator
(a,e,Mo,i,inc_t,w_peri,Omega,p,J_2)


%% Constants:

R= 6.37E6;

G=6.67384E-11; %gravity constant

M=5.972E24;    % Earth mass. (kg)

mu = G*M;

%% Formulas:


n=sqrt((mu)/a^3);% Angular speed

Omega_k = Omega - (3/2)*n*((R/p)^(2))*J_2*cos(i)*inc_t;

w_k = w_peri + (3/4)*n*((R/p)^(2))*J_2*( 5*cos(i)^2 - 1)*inc_t;

M_k= Mo + (n + (3/4)*n*((R/p)^(2))*J_2*sqrt(1 - e^2)*( 2 - 3*sin(i)^2 ) )*inc_t;% Mean
anomaly

Ek=M_k;

D=1;
while(D>=1E-8) % We need to compute the value with a certain error. The loop stops
when we reach it
  Ek1=Ek;
  Ek=M_k+e*sin(Ek1);
  D=abs(Ek-Ek1);
end
% For Ek (Eccentric Anomaly) we take the last value of the whole iteration

%for true anomaly:
sinvk=(sqrt(1-e^2)*sin(Ek))/(1-(e*cos(Ek)));
cosvk=(cos(Ek)-e)/(1-e*cos(Ek));

true_anom=angle(complex(cosvk,sinvk));

% aa = sqrt((1 + e)/(1 - e));
% bb = tan(Ek/2);
% true_anom = 2*atan(aa*bb);
% true_anom = true_anom*180/pi;

r_kepler = a*(1 - e*cos(Ek)); % Radius r.

ang_mom = sqrt(mu*a*(1 - e^2));
end
```

```
%% Inertial orbit angular velocity

function[v_ang_i] = angvel_inertial(ECI,V_ECI)


npts = size(ECI,2);
v_ang_i = zeros(3,npts);

%normalizamos el vector posicion ECI:

norm_p_eci = ((ECI(1,:).^2 + ECI(2,:).^2 + ECI(3,:).^2).^0.5);

mat_normx = repmat(norm_p_eci,3,1);

w_orb = cross(ECI,V_ECI,1)./(mat_normx.^2);

normw = ((w_orb(1,:).^2 + w_orb(2,:).^2 + w_orb(3,:).^2).^0.5);

v_ang_i(2,:) = - normw;

end



%% PLOTS

function []=
all_plots(H_rk,w_rk,angles_eu_newrk,Eclipse,ECI,timetot,dt,Tor_solar,Tor_gg,Tor_aer
o,Tor_mag)


% PLOTS MOMENTOS ANGULARES (RK-4)

figure

subplot(3,1,1)

    plot((1:timetot)*(dt/3600) , H_rk(1,:))
    title('Angular moments')
       ylabel('Hx');
       xlabel('time (h)');
subplot(3,1,2)
    plot((1:timetot)*(dt/3600),  H_rk(2,:))
       ylabel('Hy');
       xlabel('time (h)');
subplot(3,1,3)
    plot((1:timetot)*(dt/3600),  H_rk(3,:))
       ylabel('Hz');
       xlabel('time (h)');

% PLOTS VELOCIDADES ANGULARES (RK-4)

figure
```

```
subplot(3,1,1)
    plot((1:timetot)*(dt/3600) , w_rk(1,:))
    title('Angular velocities')
       ylabel('wx');
       xlabel('time (h)');
subplot(3,1,2)
    plot((1:timetot)*(dt/3600),  w_rk(2,:))
       ylabel('wy');
       xlabel('time (h)');
subplot(3,1,3)
    plot((1:timetot)*(dt/3600),  w_rk(3,:))
       ylabel('wz');
       xlabel('time (h)');

% PLOTS ANGULOS DE EULER por  RK-4(YAW, PITCH, ROLL)

figure
subplot(3,1,1)
    plot((1:timetot)*(dt/3600) , angles_eu_newrk(1,:))
       title(' Euler angles (∫)')
       ylabel('ROLL (X)');
       xlabel('time (h)');
subplot(3,1,2)
    plot((1:timetot)*(dt/3600),  angles_eu_newrk(2,:))
       ylabel('PITCH (Y)');
       xlabel('time (h)');
subplot(3,1,3)
    plot((1:timetot)*(dt/3600),  angles_eu_newrk(3,:))
       ylabel('YAW(Z)');
       xlabel('time (h)');

%PLOT ECLIPSE
figure

plot((1:timetot)*(dt/3600),Eclipse)
title('Eclipse (1 = YES)/(0 = NO)')
xlabel('Time (h)')

%PLOT TORQUE POR GRAVITY GRADIENT
figure

hold on

plot((1:timetot)*(dt/3600),Tor_gg(1,:),'b')
title('Gravity Gradient Torque ')
xlabel('Time (h)')
ylabel(' Disturbance Moment (N∑m)')

plot((1:timetot)*(dt/3600),Tor_gg(2,:),'r')

plot((1:timetot)*(dt/3600),Tor_gg(3,:),'k')
legend({'X-Torque','Y-Torque','Z-Torque'})
hold off

%PLOT TORQUE SOLAR
```

```
figure

hold on

plot((1:timetot)*(dt/3600),Tor_solar(1,:),'b')
title('Solar Pressure Torque ')
xlabel('Time (h)')
ylabel(' Disturbance Moment (N∑m)')

plot((1:timetot)*(dt/3600),Tor_solar(2,:),'r')

plot((1:timetot)*(dt/3600),Tor_solar(3,:),'g')
legend({'X-Torque','Y-Torque','Z-Torque'})
hold off

%PLOT TORQUE AERODINAMICO

figure

hold on

plot((1:timetot)*(dt/3600),Tor_aero(1,:),'b')
title('Aerodynamic Torque ')
xlabel('Time (h)')
ylabel(' Disturbance Moment (N∑m)')

plot((1:timetot)*(dt/3600),Tor_aero(2,:),'r')

plot((1:timetot)*(dt/3600),Tor_aero(3,:),'g')
legend({'X-Torque','Y-Torque','Z-Torque'})
hold off

%PLOT TORQUE MAGNETICO

figure

hold on

plot((1:timetot)*(dt/3600),Tor_mag(1,:),'b')
title('Magnetic Torque ')
xlabel('Time (h)')
ylabel(' Disturbance Moment (N∑m)')

plot((1:timetot)*(dt/3600),Tor_mag(2,:),'r')

plot((1:timetot)*(dt/3600),Tor_mag(3,:),'g')
legend({'X-Torque','Y-Torque','Z-Torque'})
hold off

%PLOT ORBITA
figure
plot(ECI(1,1:15000),ECI(2,1:15000),'o')
title('Orbit')
xlabel('X (m)')
ylabel('Y (m)')
```

```
end
```

Main program:

```
clc
clear
close all
%% Attitude Representation of a CubeSat

%Nuestro satelite tendra 6 caras (por lo que la matriz tendr· 6 columnas
%con cada cara respectivamente y tendra 3 filas representando x, y ,z
%respectivamente:
        %Cara 1 --> +x ()
        %Cara 2 --> -x
        %Cara 3 --> +y (Sur)
        %Cara 4 --> -y
        %Cara 5 --> +z (Nadir)
        %Cara 6 --> -z

% Ejes cuerpo:

%Supongaamos que el eje que apunta hacia Nadir es el eje +z, el que apunta
%al sur es el eje +y:

Caras = [ 1 -1 0 0 0 0;  % x
        0 0 1 -1 0 0;  % y
        0 0 0 0 1 -1]; % z

Caras = Caras';

Areas = [0.01 0 0;
        0.01 0 0;
        0 0.01 0;
        0 0.01 0;
        0 0 0.01;
        0 0 0.01]; % m^2

%%                      DATA SATELLITE

m = 1.33;           % kg

% inertia = [ 0.018 0 0;
%           0 0.018 0;
%           0 0 0.006]; %kg∑m^2

inertia = [ 0.00221 0 0;
          0 0.00221 0;
          0 0 0.00221]; %kg∑m^2

% inertia = [2000 300 -200;
```

```
%          300 4000 -100;
%          -200 -100 1000];

inv_inertia = inv(inertia);

r_geo_cm = [ 0.001;
                0;
           -0.009];    % distance between the geographical center and
                       % the center of mass (m).

%%                    ORBITAL PARAMETERS



R_earth = 6.37E6;       % m

altitude = 650E+3;      % m

r = altitude + R_earth; % m

ro = 0.227E-13;         % Kg/m^3 (density at 650000 Km of altitude)


%ORBIT PARAMETERS (KEPLER)

J_2 = 1.083E-3;       %Harmonic cofficient

i_orbit = 20*pi/180 ;  %Orbital inclination (rad)

w_peri = 97*pi/180;  %Argument of periapsis (rad)

Mo = 76*pi/180;      %Mean anomaly (rad)

e = 0.5;      %Eccentricity

Omega = 131*pi/180;  %Longitude ascending node (rad)

a = r;  %Semimajor axis (m)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


mag_dip = 7.9577E15;

p = a*(1 - e^2);

w_earth = 7.2921E-5;    % rad/s

w_e_i =[0;
        0;
        w_earth];       % vel. de rotaciÛn Tierra (Inertial Frame)

CD = 2; % aerodynamic drag coefficient
```

```
Ps = 4.56E-6;  % (N/m^2) solar radiation pressure near Earth (at 1 AU)

epsilon = 0.21;         % reflectivity effect factor

M_resid = [0;
     0.0005;
          0];         % residual dipole vector (A∑m^2)

mu = 3.986E+14;         % standard gravitational parameter

Ts = 0;                 % Days from the fist day of spring (for elevation)



%%                          SIMULATION

num_orb = 5;                    % N˙mero de Ûrbitas para la simulaciÛn?

T_orbital = 2*pi*sqrt((r^3)/mu);    % Periodo orbital (s)

t_sim  = num_orb*T_orbital;         %Tiempo de simulacion. (s)

dt     = 1;                         %Tiempo de muestreo. (s)

h=dt;

dt_dias = dt/(3600*24);             % Para el incremento del Ts (posiciÛn
                                    % sol varìa a medida que pasa el tiempo
                                    % de sim.) (dias)

timetot     = round(t_sim / dt); % Longitud que tendra el vector de velocidades
                                 % debido al dt.

%%  Propagator (Kepler -- ECI)

%En este primer caso,como empezamos la simulaciÛn, el inc_dt es 0 y n es 1.

for i = 1:timetot

%[true_anom,ang_mom,M_k,Ek,r_kepler] = Kep_propagator (a,e,Mo,dt);           %
function

[true_anom,ang_mom,M_k,Ek,r_kepler,w_k,Omega_k] = J2_propagator
(a,e,Mo,i,dt,w_peri,Omega,p,J_2);


%we save all data in Kepler_Data Struct

Kepler_Data.w_peri(i) = w_k;

Kepler_Data.Omega(i) = Omega_k;

Kepler_Data.TrueAnom(i) = true_anom;
```

```
Kepler_Data.Angular_mom(i) = ang_mom;

Kepler_Data.Mean_Anom(i) = M_k;

Kepler_Data.Ecc_Anom(i) = Ek;

Kepler_Data.r(i) = r_kepler; %will be constant if its circular orbit

% FEEDBACK

w_peri = Kepler_Data.w_peri(i);

Omega = Kepler_Data.Omega(i);

Mo = Kepler_Data.Mean_Anom(i);

%Si queremos pasar a coordenadas ECI:

[X,Y,Z,X_vel,Y_vel,Z_vel] =
Kepler_To_ECI(w_k,i_orbit,Omega_k,r_kepler,true_anom,ang_mom,p,e);   %function

% Guardamos las posiciones (vector) en una matriz ECI.

ECI(1,i) = X;
ECI(2,i) = Y;
ECI(3,i) = Z;

% Guardamos las vel (vector) en ECI en una matriz V_ECI. (Lo usaremos para
calcular la
% F aerodinamica).

V_ECI(1,i) = X_vel;
V_ECI(2,i) = Y_vel;
V_ECI(3,i) = Z_vel;

% VECTOR vr para el calculo de fuerza aerodin·mica.

v_r(:,i) = V_ECI(:,i) - cross( w_e_i, ECI(:,i));

end

%% Geomagnetic Field Model

t = 0;

for i = 1:timetot

   B_earth(:,i) = (mag_dip/(Kepler_Data.r(i))^3)*[ cos(w_earth*t)*sin(i_orbit);
                                                   -cos(i_orbit);
                                                   2*sin(w_earth*t)*sin(i_orbit)];

   t = t + dt;
end

%% INITIAL CONDITIONS
```

```
%primera fila --> psi (roll)
%segunda fila --> theta (pitch)
%tercera fila --> phi (yaw)

% A_euler = [rand*360 * pi/180;
%            rand*360 * pi/180;
%            rand*360 * pi/180]; %Trabajaremos en radianes

%ANGLES

A_euler = [0 * pi/180;
           0 * pi/180;
           0 * pi/180]; %Trabajaremos en radianes

angles_eu_rk(:,1) = A_euler;

%Angular VEL

 %primera fila --> wx
 %segunda fila --> wy
 %tercera fila --> wz

v_ang_i = angvel_inertial(ECI,V_ECI);

ang_vel     = [0;
               0;
               0]; %Trabajaremos en rad/s

w_rk(:,1) = ang_vel + v_ang_i(:,1);

%Ang Moment.

H_rk(:,1) = inertia*w_rk(:,1);

% torque = [0;
%           3E-7;
%           0];

%% For gravity gradient:

%Transformations

R_OP_I = ECI_To_OrbitPlane(i_orbit,Kepler_Data.Omega(1));

R_LocalOrbit_OP= OrbitP_To_LocalOrbit(Kepler_Data.TrueAnom(1));

R_Body_LocalOrbit =
OrbitP_To_Body(angles_eu_rk(1,1),angles_eu_rk(2,1),angles_eu_rk(3,1));

%%%%
vec_nad = [0;0;1];

u_nadir = R_Body_LocalOrbit*vec_nad;
```

```
Tor_gg(:,1) = ((3*mu)/Kepler_Data.r(1)^(3))*cross(u_nadir,(inertia*u_nadir));

%% For Solar pressure Force

%S_i(:,1) = [1;      This is for first day of SPRING
%           0;
%           0];

% But if we are not in the first day:

lamda_1 = ((2*pi)/365)*Ts;

Es_1    = ((23*pi)/180)*sin(lamda_1);

S_I(:,1) =  [cos(Es_1)*cos(lamda_1); %Kristiansen et al proposal
          sin(lamda_1);
          sin(Es_1)*cos(lamda_1)];

U_shadow(:,1) = -S_I(:,1); % Eclipse vector


S_B(:,1) = R_Body_LocalOrbit*R_LocalOrbit_OP*R_OP_I*S_I(:,1); %Sun vector in
bodyframe from ECI.


nightside(1) = dot(U_shadow(:,1),ECI(:,1));

shadow_cylinder(1) = abs(R_earth*norm(cross(U_shadow(:,1),ECI(:,1))));

%Calculo de angulos para cada cara respecto al Sun vector en body frame:

Sum_Sun_Force(:,1) = zeros(3,1);

Sum_Aero_Force(:,1) = zeros(3,1);

n = 1;
while (n <= length(Caras))


%angle between normal of the faces and sun vector

CosTheta(n,1) = dot(S_B(:,1),Caras(n,:))/(norm(S_B(:,1))*norm(Caras(n,:)));
Theta(n,1) = acosd(CosTheta(n,1));

if( abs(Theta(n,1)) >= 90)

   CosTheta(n,1) = 0;
end

aaa = (1-epsilon)*S_B(:,1)';

bbb = 2*epsilon*CosTheta(n,1)*Caras(n,:);

%USO UNA STRUCT CON 6 SECCIONES (QUE SER¡N LAS CARAS) PORQUE LA
FUERZA
```

```
%SOLAR EN CADA CARA TENDR¡  UN VECTOR DE 3 POSICIONES.

F_solar(n).Faces(:,1) = (-Ps*sum(Areas(n,:))*CosTheta(n,1)*( aaa + bbb))';

Sum_Sun_Force(:,1) = Sum_Sun_Force(:,1) + F_solar(n).Faces(:,1);


if ((nightside(1) > 0) && (shadow_cylinder(1) <= R_earth))

   Tor_solar (:,1) = 0;
   Eclipse(1)=1;
else

   Tor_solar (:,1) = cross (r_geo_cm,Sum_Sun_Force(:,1));
   Eclipse(1)=0;
end

%% Aerodynamic Torque

%Procederemos de la misma forma que con el Solar pressure torque, si
%queremos saber que caras intervienen en el calculo de drag hemos de
%encontrar el ·ngulo entre el vector velocidad y la normal de las caras.

v_r_bf(:,1) =  R_Body_LocalOrbit*R_LocalOrbit_OP*R_OP_I*v_r(:,1);

CosTheta_aero(n,1) = dot(v_r_bf(:,1),Caras(n,:))/(norm(v_r_bf(:,1))*norm(Caras(n,:)));

Theta_aero(n,1) = acosd(CosTheta_aero(n,1));

if( abs(Theta_aero(n,1)) >= 90)

    F_aero(n).Faces(:,1) = zeros(3,1);

else

    F_aero(n).Faces(:,1) = (-
0.5)*CD*sum(Areas(n,:))*ro*(norm(v_r(:,1))^(2))*(v_r(:,1)/norm(v_r(:,1)));

end

   Sum_Aero_Force(:,1) = Sum_Aero_Force(:,1) + F_aero(n).Faces(:,1);

   Tor_aero (:,1) = cross (r_geo_cm,Sum_Aero_Force(:,1));

n = n + 1;

end

Ts = Ts + dt_dias;

%% Magnetic Torque

   Tor_mag(:,1) = cross( M_resid , B_earth(:,1));

%% General torque
```

```
Torque(:,1) = Tor_gg(:,1) + Tor_solar(:,1) + Tor_aero(:,1) + Tor_mag(:,1);

format long

%% MAIN LOOP


for p=2:timetot


    roll = angles_eu_rk(1,p-1);
    pitch = angles_eu_rk(2,p-1);
    yaw = angles_eu_rk(3,p-1);


%%                          TORQUES

%% GRAVITY GRADIENT TORQUE

r = Kepler_Data.r(p);

u_nadir = [-sin(pitch);
           cos(pitch)*sin(roll);
           cos(pitch)*cos(roll)];

Tor_gg(:,p) = cross((((3*mu)/r^(3))*u_nadir),(inertia*u_nadir));

%Torgg = 0;
%% SOLAR PRESSURE TORQUE

lamda = ((2*pi)/365)*Ts;

Es    = ((23*pi)/180)*sin(lamda);

S_I(:,p) = [cos(Es)*cos(lamda);
            sin(lamda);
            sin(Es)*cos(lamda)];

U_shadow(:,p) = -S_I(:,p); % Eclipse vector

%hemos de mirar antes de hacer el calculo de la fuerza por radiacion solar,
%si nuestro satelite se encuentra en la zona de noche y dentro de la sombra
%en nuestro modelo de cilindro. DEBE SATISFACER DOS CONDICIONES:

nightside(p) = dot(U_shadow(:,p),ECI(:,p));

shadow_cylinder(p) = abs(norm(cross(U_shadow(:,p),ECI(:,p))));

% Como el vector posicion del sol esta en los ejes incerciales, aplicamos
% la rotaciÛn para tenerlo en los ejes cuerpo.

R_OP_I = ECI_To_OrbitPlane(i_orbit,Kepler_Data.Omega(i));
```

```
R_LocalOrbit_OP= OrbitP_To_LocalOrbit(Kepler_Data.TrueAnom(p));

R_Body_LocalOrbit = OrbitP_To_Body(roll,pitch,yaw);

S_B(:,p) = R_Body_LocalOrbit*R_LocalOrbit_OP*R_OP_I*S_I(:,p);

%Once we have the Sun vector in body frame we compute the angle between
%sun vector and normal vector of each face.
%Also we should delete the contribution of those faces that have more than
%90 degrees. It means that sun is not illuminating this face.

n = 1;

% Cada iteracion se actualiza a 0 el sumador porque es para cada situaciÛn.

Sum_Sun_Force(:,p) = zeros(3,1);

Sum_Aero_Force(:,p) = zeros(3,1);

while (n <= length(Caras))

CosTheta(n,p) = dot(S_B(:,p),Caras(n,:))/(norm(S_B(:,p))*norm(Caras(n,:)));

Theta(n,p) = acosd(CosTheta(n,p));

%De una vez aÒado las condiciones del  MODELO DE ECLIPSE.
if(Theta(n,p) >= 90)

   CosTheta(n,p) = 0;

end

% Una vez visto que caras son las que estan expuestas al sol, calculamos la
% Fuerza de presion solar.

%USO UNA STRUCT CON 6 SECCIONES (QUE SER¡N LAS CARAS) PORQUE LA
FUERZA
%SOLAR EN CADA CARA TENDR¡  UN VECTOR DE 3 POSICIONES.

aa = (1-epsilon)*S_B(:,p)';
bb = 2*epsilon*CosTheta(n,p)*(Caras(n,:));

F_solar(n).Faces(:,p) = (-Ps*sum(Areas(n,:))*CosTheta(n,p)*(aa + bb))';

%Sumatorio de todas las fuerzas que provienen de las diferentes caras que
%son iluminadas por el sol.

Sum_Sun_Force(:,p) = Sum_Sun_Force(:,p) + F_solar(n).Faces(:,p);

% Ahora encontramos apartir de la fÛrmula general, el torque solar.

if ((nightside(p) > 0) && (shadow_cylinder(p)<= (R_earth)))

    Tor_solar (:,p) = zeros(3,1);
```

```
   Eclipse(p) = 1;

else

   Tor_solar (:,p) = cross (r_geo_cm,Sum_Sun_Force(:,p));
   Eclipse(p) = 0;

end

%% Aerodynamic Torque

%Procederemos de la misma forma que con el Solar pressure torque, si
%queremos saber que caras intervienen en el calculo de drag hemos de
%encontrar el ·ngulo entre el vector velocidad y la normal de las caras.

v_r_bf(:,p) =  R_Body_LocalOrbit*R_LocalOrbit_OP*R_OP_I*v_r(:,p);

CosTheta_aero(n,p) = dot(v_r_bf(:,p),Caras(n,:))/(norm(v_r_bf(:,p))*norm(Caras(n,:)));

Theta_aero(n,p) = acosd(CosTheta_aero(n,p));

if( abs(Theta_aero(n,p)) >= 90)

    F_aero(n).Faces(:,p) = zeros(3,1);

else

    F_aero(n).Faces(:,p) = -0.5*CD*sum(Areas(n,:))*ro*norm(v_r(:,p))*v_r(:,p);

end

   Sum_Aero_Force(:,p) = Sum_Aero_Force(:,p) + F_aero(n).Faces(:,p);

   Tor_aero (:,p) = cross (r_geo_cm,Sum_Aero_Force(:,p));

n = n + 1;

end

%% Magnetic Torque


   Tor_mag(:,p) = cross( M_resid , B_earth(:,p));


%% General Torque

Torque(:,p) = Tor_gg(:,p) + Tor_solar(:,p) + Tor_aero(:,p) + Tor_mag(:,p);
% Torque(:,p) = zeros(3,1);

%Escribimos las variables del RK-4
   w_i = w_rk(:,p-1);
   k1 =inv_inertia*(Torque(:,p) - cross(w_i, inertia*w_i));
   k2 =inv_inertia*(Torque(:,p) - cross((w_i + h*k1/2), inertia*(w_i + h*k1/2)));
   k3 =inv_inertia*(Torque(:,p) - cross((w_i + h*k2/2), inertia*(w_i + h*k2/2)));
```

```
    k4 =inv_inertia*(Torque(:,p) - cross((w_i + h*k3), inertia*(w_i+h*k3)));

    w_rk(:,p) = w_i + h/6 * ( k1 + 2*k2 + 2*k3 + k4);

    % Matriz que transforma desde la velocidad angular w a las vel
    % angulares de pitch, roll y yaw.

    H_rotation =[1 sin(roll)*tan(pitch) cos(roll)*tan(pitch);
        0 cos(roll) -sin(roll);
        0 sin(roll)/cos(pitch) cos(roll)/cos(pitch)];

    angles_eu_rk(:,p) = angles_eu_rk(:,p-1) + dt*H_rotation*w_rk(:,p);

    H_rk(:,p) = inertia*w_rk(:,p);

    Ts = Ts + dt_dias;

end

% HACEMOS LA CORRECCI”N DE LOS ¡NGULOS

angles_eu_newrk = [];

for (f=1:3)

  for (k=1:timetot)
      if ((angles_eu_rk(f,k)*(180/pi)) > 180 || (angles_eu_rk(f,k)*(180/pi)) < -180 )

          % De una vez lo pasamos a grados, en lugar de radianes.
          angles_eu_newrk(f,k) = mod((angles_eu_rk(f,k)*(180/pi)),360);

      else
          angles_eu_newrk(f,k) = (angles_eu_rk(f,k)*(180/pi));

      end
  end
end

%% Plots

all_plots(H_rk,w_rk,angles_eu_newrk,Eclipse,ECI,timetot,dt,Tor_solar,Tor_gg,Tor_aer
o,Tor_mag);
```